



Script Syntax and Chart Functions Guide

Qlik® Sense

1.0.3

Copyright © 1993-2015 QlikTech International AB. All rights reserved.



Copyright © 1993-2015 QlikTech International AB. All rights reserved.

Qlik®, QlikTech®, Qlik® Sense, QlikView®, Sense™ and the Qlik logo are trademarks which have been registered in multiple countries or otherwise used as trademarks by QlikTech International AB. Other trademarks referenced herein are the trademarks of their respective owners.

1 What is Qlik Sense?	11
1.1 What can you do in Qlik Sense?	11
1.2 How does Qlik Sense work?	11
The app model	11
The associative experience	11
Collaboration and mobility	11
1.3 How can you deploy Qlik Sense?	11
Qlik Sense Desktop	12
Qlik Sense Server	12
1.4 How to administer and manage a Qlik Sense site	12
1.5 Extend Qlik Sense and adapt it for your own purposes	12
Building extensions and mashups	12
Building clients	12
Building server tools	12
Connecting to other data sources	12
2 Script syntax	13
2.1 Introduction to script syntax	13
2.2 What is Backus-Naur formalism?	13
2.3 Script statements and keywords	14
Script control statements	14
Script prefixes	25
Script regular statements	50
2.4 Script variables	98
Variable calculation	99
System variables	100
Value handling variables	104
Number interpretation variables	106
Direct Discovery variables	113
Error variables	116
2.5 Script expressions	119
3 Visualization expressions	121
3.1 Defining the aggregation scope	121
3.2 Analyzing sets of data - Set analysis	123
Building a set expression	124
Set identifiers	125
Set operators	126
Set modifiers	127
Syntax for sets	131
3.3 Syntax	132
General syntax for chart expressions	132
General syntax for aggregations	133
4 Operators	134
4.1 Bit operators	134
4.2 Logical operators	135

4.3 Numeric operators	135
4.4 Relational operators	135
4.5 String operators	137
5 Functions in scripts and chart expressions	138
5.1 Aggregation functions	138
Using aggregation functions in a data load script	138
Using aggregation functions in chart expressions	138
Aggr - chart function	138
Basic aggregation functions	141
Counter aggregation functions	160
Financial aggregation functions	174
Statistical aggregation functions	183
Statistical test functions	229
String aggregation functions	317
Synthetic dimension functions	326
Nested aggregations	328
5.2 Color functions	329
ARGB	330
RGB	330
HSL	331
5.3 Conditional functions	332
Conditional functions overview	332
alt	333
class	333
if	334
match	334
mixmatch	335
pick	335
wildmatch	335
5.4 Counter functions	336
Counter functions overview	336
autonumber	337
autonumberhash128	338
autonumberhash256	338
fieldvaluecount	338
IterNo	338
RecNo	339
RowNo	339
RowNo - chart function	340
5.5 Date and time functions	342
Date and time functions overview	342
addmonths	350
addyears	351
age	352
converttolocaltime	352

Contents

day	354
dayend	354
daylightsaving	355
dayname	355
daynumberofquarter	356
daynumberofyear	356
daystart	357
firstworkdate	358
GMT	358
hour	358
inday	359
indaytotime	359
inlunarweek	360
inlunarweektoday	361
inmonth	361
inmonths	362
inmonthstodate	363
inmonthtoday	363
inquarter	364
inquartertoday	365
inweek	365
inweektoday	366
inyear	367
inyeartoday	368
lastworkdate	368
localtime	369
lunarweekend	369
lunarweekname	370
lunarweekstart	371
makedate	371
maketime	372
makeweekdate	373
minute	373
month	374
monthend	374
monthname	375
monthsend	375
monthsname	376
monthsstart	377
monthstart	377
networkdays	378
now	378
quarterend	379
quartername	379
quarterstart	380
second	381

setdateyear	381
setdateyearmonth	382
timezone	382
today	382
UTC	383
week	383
weekday	383
weekend	384
weekname	385
weekstart	386
weekyear	387
year	387
yearend	388
yearname	388
yearstart	389
yeartodate	390
5.6 Exponential and logarithmic functions	391
5.7 Field functions	392
Count functions	392
Field and selection functions	392
GetAlternativeCount - chart function	393
GetCurrentSelections - chart function	394
GetExcludedCount - chart function	395
GetFieldSelections - chart function	396
GetNotSelectedCount - chart function	398
GetPossibleCount - chart function	398
GetSelectedCount - chart function	399
5.8 File functions	400
File functions overview	400
Attribute	402
ConnectionString	409
FileName	410
FileDir	410
FileExtension	410
FileName	411
FilePath	411
FileSize	411
FileTime	412
GetFolderPath	413
QvdCreateTime	413
QvdFieldName	414
QvdNoOfFields	415
QvdNoOfRecords	416
QvdTableName	416
5.9 Financial functions	417

Financial functions overview	417
Black and Scholes	418
FV	419
nPer	419
Pmt	420
PV	421
Rate	422
5.10 Formatting functions	423
Formatting functions overview	423
Date	424
Dual	425
Interval	425
Num	426
Money	427
Time	427
Timestamp	428
5.11 General numeric functions in charts	428
BitCount	430
Ceil	431
Combin	431
Div	432
Even	433
Fabs	433
Fact	434
Floor	434
Fmod	435
Frac	435
Mod	436
Odd	437
Permut	437
Round	438
Sign	438
5.12 Geographical functions	439
Geographical functions overview	439
5.13 Interpretation functions	440
Interpretation functions overview	441
Date#	442
Interval#	442
Money#	443
Num#	444
Text	444
Time#	445
Timestamp#	445
5.14 Inter-record functions	446
Row functions	446

Column functions	447
Field functions	447
Inter-record functions in the data load script	448
Above - chart function	449
Bottom - chart function	453
Below - chart function	456
Column - chart function	459
Dimensionality - chart function	461
Exists	462
FieldIndex	462
FieldValue	463
FieldValueCount	465
LookUp	466
NoOfRows - chart function	466
Peek	467
Previous	468
Top - chart function	468
5.15 Logical functions	472
5.16 Mapping functions	472
Mapping functions overview	472
ApplyMap	472
MapSubstring	473
5.17 Mathematical constants and parameter-free functions	474
5.18 NULL functions	475
NULL functions overview	475
IsNull	475
NULL	475
5.19 Range functions	476
Basic range functions	476
Counter range functions	477
Statistical range functions	477
RangeAvg	478
RangeCorrel	479
RangeCount	480
RangeFractile	481
RangeIRR	483
RangeKurtosis	483
RangeMax	484
RangeMaxString	486
RangeMin	487
RangeMinString	488
RangeMissingCount	489
RangeMode	491
RangeNPV	493
RangeNullCount	494

RangeNumericCount	495
RangeOnly	496
RangeSkew	498
RangeStdev	499
RangeSum	500
RangeTextCount	502
RangeXIRR	503
RangeXNPV	504
5.20 Ranking functions in charts	504
Rank - chart function	505
VRank - chart function	508
5.21 Statistical distribution functions	508
Statistical distribution functions overview	508
CHIDIST	509
CHIINV	510
FDIST	510
FINV	511
NORMDIST	512
NORMINV	512
TDIST	513
TINV	514
5.22 String functions	514
String functions overview	515
Capitalize	518
Chr	518
Evaluate	518
FindOneOf	519
Hash128	519
Hash160	519
Hash256	520
Index	520
KeepChar	520
Left	521
Len	521
Lower	521
LTrim	522
Mid	522
Ord	523
PurgeChar	523
Repeat	523
Replace	523
Right	524
RTrim	524
SubField	525
SubStringCount	525

TextBetween	525
Trim	526
Upper	526
5.23 System functions	526
System functions overview	526
GetExtendedProperty - chart function	528
GetObjectField - chart function	529
QlikViewVersion	529
5.24 Table functions	529
Table functions overview	529
FieldName	530
FieldNumber	530
NoOfFields	531
NoOfRows	531
5.25 Trigonometric and hyperbolic functions	531
5 File system access restriction	533
5.26 Limitations in standard mode	533
System variables	533
Regular script statements	534
Script control statements	535
File functions	535
System functions	537
5.27 Disabling standard mode	538
Qlik Sense	538
Qlik Sense Desktop	538

1 What is Qlik Sense?

Qlik Sense is a platform for data analysis. With Qlik Sense you can analyze data and make data discoveries on your own. You can share knowledge and analyze data in groups and across organizations. Qlik Sense lets you ask and answer your own questions and follow your own paths to insight. Qlik Sense enables you and your colleagues to reach decisions collaboratively.

1.1 What can you do in Qlik Sense?

Most Business Intelligence (BI) products can help you answer questions that are understood in advance. But what about your follow-up questions? The ones that come after someone reads your report or sees your visualization? With the Qlik Sense associative experience, you can answer question after question after question, moving along your own path to insight. With Qlik Sense you can explore your data freely, with just clicks, learning at each step along the way and coming up with next steps based on earlier findings.

1.2 How does Qlik Sense work?

Qlik Sense generates views of information on the fly for you. Qlik Sense does not require predefined and static reports or you being dependent on other users – you just click and learn. Every time you click, Qlik Sense instantly responds, updating every Qlik Sense visualization and view in the app with a newly calculated set of data and visualizations specific to your selections.

The app model

Instead of deploying and managing huge business applications, you can create your own Qlik Sense apps that you can reuse, modify and share with others. The app model helps you ask and answer the next question on your own, without having to go back to an expert for a new report or visualization.

The associative experience

Qlik Sense automatically manages all the relationships in the data and presents information to you using a **green/white/gray** metaphor. Selections are highlighted in green, associated data is represented in white, and excluded (unassociated) data appears in gray. This instant feedback enables you to think of new questions and continue to explore and discover.

Collaboration and mobility

Qlik Sense further enables you to collaborate with colleagues no matter when and where they are located. All Qlik Sense capabilities, including the associative experience and collaboration, are available on mobile devices. With Qlik Sense, you can ask and answer your questions and follow-up questions, with your colleagues, wherever you are.

1.3 How can you deploy Qlik Sense?

There are two versions of Qlik Sense to deploy, Qlik Sense Desktop and Qlik Sense Server.

Qlik Sense Desktop

This is an easy-to-install single user version that is typically installed on a local computer.

Qlik Sense Server

This version is used to deploy Qlik Sense sites. A site is a collection of one or more server machines connected to a common logical repository or central node.

1.4 How to administer and manage a Qlik Sense site

With the Qlik Management Console you can configure, manage and monitor Qlik Sense sites in an easy and intuitive way. You can manage licenses, access and security rules, configure nodes and data source connections and synchronize content and users among many other activities and resources.

1.5 Extend Qlik Sense and adapt it for your own purposes

Qlik Sense provides you with flexible APIs and SDKs to develop your own extensions and adapt and integrate Qlik Sense for different purposes, such as:

Building extensions and mashups

Here you can do web development using JavaScript to build extensions that are custom visualization in Qlik Sense apps, or you use a mashups APIs to build websites with Qlik Sense content.

Building clients

You can build clients in .NET and embed Qlik Sense objects in your own applications. You can also build native clients in any programming language that can handle WebSocket communication by using the Qlik Sense client protocol.

Building server tools

With service and user directory APIs you can build your own tool to administer and manage Qlik Sense sites.

Connecting to other data sources

Create Qlik Sense connectors to retrieve data from custom data sources.

2 Script syntax

2.1 Introduction to script syntax

In the script, the name of the data source, the names of the tables and the names of the fields included in the logic are defined. Furthermore, the fields in the access rights definition are defined in the script.

The script consists of a number of statements that are executed consecutively.

The Qlik Sense command line syntax and script syntax are described in a notation called Backus-Naur Formalism, or BNF code.

The first lines of code are already generated when a new Qlik Sense file is created. The default values of these number interpretation variables are derived from the regional settings of the OS.

In the script the name of the data source, the names of the tables and the names of the fields included in the logic are defined. The script consists of a number of script statements and keywords that are executed consecutively.

For a table file with commas, tabs or semicolons as delimiter, the **LOAD**-statement may be used. By default the **LOAD**-statement will load all fields of the file.

A general database must be accessed through Microsoft ODBC. Here standard SQL statements are used. The SQL syntax accepted differs between different ODBC drivers.

All script statements must end with a semicolon, ";".

A detailed description of script syntax can be accessed via the topics in this section.

2.2 What is Backus-Naur formalism?

The Qlik Sense command line syntax and script syntax are described in a notation called Backus-Naur formalism, or BNF code. Here follows a short description of the BNF code:

The following symbols should be interpreted like this:

	Logical or: the symbol on either side can be used.
()	Brackets defining precedence: used for structuring the BNF syntax.
[]	Square brackets: enclosed items are optional.
{ }	Braces: enclosed items may be repeated zero or more times.
Symbol	A non-terminal syntactic category: can be divided further into other symbols, e.g. compounds of the above, other non-terminal symbols, text strings, etc.
::=	Marks the beginning of a block that defines a symbol.
LOAD	A terminal symbol consisting of a text string. Should be written as it is into the script.

All terminal symbols are printed in a **bold face** font. E.g. "(" should be interpreted as a bracket defining precedence, whereas "(" should be interpreted as a character that should be printed in the script.

Example:

The description of the alias statement is:

```
alias fieldname as aliasname { , fieldname as aliasname }
```

This should be interpreted as the text string "alias", followed by an arbitrary field name, followed by the text string "as", followed by an arbitrary alias name. Any number of additional combinations of "fieldname as alias" may be given, separated by commas.

The following statements are correct:

```
alias a as first;  
alias a as first, b as second;  
alias a as first, b as second, c as third;
```

The following statements are not correct:

```
alias a as first b as second;  
alias a as first { , b as second };
```

2.3 Script statements and keywords

The Qlik Sense script consists of a number of statements. A statement can be either a regular script statement or a script control statement. Certain statements can be preceded by prefixes.

Regular statements are typically used for manipulating data in one way or another. These statements may be written over any number of lines in the script and must always be terminated by a semicolon, ";".

Control statements are typically used for controlling the flow of the script execution. Each clause of a control statement must be kept inside one script line and may be terminated by a semicolon or the end-of-line.

Prefixes may be applied to applicable regular statements but never to control statements. The **when** and **unless** prefixes can however be used as suffixes to a few specific control statement clauses.

In the next subchapter, an alphabetical listing of all script statements, control statements and prefixes, are found.

All script keywords can be typed with any combination of lower case and upper case characters. Field and variable names used in the statements are however case sensitive.

Script control statements

The Qlik Sense script consists of a number of statements. A statement can be either a regular script statement or a script control statement.

Control statements are typically used for controlling the flow of the script execution. Each clause of a control statement must be kept inside one script line and may be terminated by semicolon or end-of-line.

Prefixes are never applied to control statements, with the exceptions of the prefixes **when** and **unless** which may be used with a few specific control statements.

All script keywords can be typed with any combination of lower case and upper case characters.

Script control statements overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Call

The **call** control statement calls a subroutine which must be defined by a previous **sub** statement.

```
Call name ( [ paramlist ] )
```

Do..loop

The **do..loop** control statement is a script iteration construct which executes one or several statements until a logical condition is met.

```
Do..loop [ ( while | until ) condition ] [statements]  
[exit do [ ( when | unless ) condition ] [statements]  
loop [ ( while | until ) condition ]
```

Exit script

This control statement stops script execution. It may be inserted anywhere in the script.

```
Exit script [ (when | unless) condition ]
```

For each ..next

The **for each..next** control statement is a script iteration construct which executes one or several statements for each value in a comma separated list. The statements inside the loop enclosed by **for** and **next** will be executed for each value of the list.

```
For each..next var in list  
[statements]  
[exit for [ ( when | unless ) condition ]  
[statements]  
next [var]
```

For..next

The **for..next** control statement is a script iteration construct with a counter. The statements inside the loop enclosed by **for** and **next** will be executed for each value of the counter variable between specified low and high limits.

```
For..next counter = expr1 to expr2 [ stepexpr3 ]  
[statements]  
[exit for [ ( when | unless ) condition ]  
[statements]  
Next [counter]
```

If..then

The **if..then** control statement is a script selection construct forcing the script execution to follow different paths depending on one or several logical conditions.



Since the **if..then** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its four possible clauses (**if..then**, **elseif..then**, **else** and **end if**) must not cross a line boundary.

```
If..then..elseif..else..end if condition then
  [ statements ]
{ elseif condition then
  [ statements ] }
[ else
  [ statements ] ]
end if
```

Sub

The **sub..end sub** control statement defines a subroutine which can be called upon from a **call** statement.

```
Sub..end sub name [ ( paramlist ) ] statements end sub
```

Switch

The **switch** control statement is a script selection construct forcing the script execution to follow different paths, depending on the value of an expression.

```
Switch..case..default..end switch expression {case valuelist [ statements
]} [default statements] end switch
```

Call

The **call** control statement calls a subroutine which must be defined by a previous **sub** statement.

Syntax:

```
Call name ( [ paramlist ] )
```

Arguments:

Argument	Description
name	The name of the subroutine.
paramlist	A comma separated list of the actual parameters to be sent to the subroutine. Each item in the list may be a field name, a variable or an arbitrary expression.

The subroutine called by a **call** statement must be defined by a **sub** encountered earlier during script execution.

Parameters are copied into the subroutine and, if the parameter in the **call** statement is a variable and not an expression, copied back out again upon exiting the subroutine.

Limitations:

Since the **call** statement is a control statement and as such is ended with either a semicolon or end-of-line, it must not cross a line boundary.

Example 1:

```
// Example 1
Sub INCR (I,J)
    I = I + 1
    Exit Sub when I < 10
    J = J + 1
End Sub
Call INCR (X,Y)
```

Example 2:

```
// Example 2 - List all QV related files on disk
sub DoDir (Root)
    For Each Ext in 'qvw', 'qvo', 'qvs', 'qvt', 'qvd', 'qvc', 'qvf'
        For Each File in filelist (Root&'\'*' &Ext)
            LOAD
                '$(File)' as Name, FileSize( '$(File)' ) as
                Size, FileTime( '$(File)' ) as FileTime
                autogenerate 1;
        Next File
    Next Ext
    For Each Dir in dirlist (Root&'\'*' )
        Call DoDir (Dir)
    Next Dir
End Sub
Call DoDir ('C:')
```

Do..loop

The **do..loop** control statement is a script iteration construct which executes one or several statements until a logical condition is met.

Syntax:

```
Do [ ( while | until ) condition ] [statements]
[exit do [ ( when | unless ) condition ] [statements]
loop[ ( while | until ) condition ]
```



Since the **do..loop** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its three possible clauses (**do**, **exit do** and **loop**) must not cross a line boundary.

Arguments:

Argument	Description
condition	A logical expression evaluating to True or False.
statements	Any group of one or more Qlik Sense script statements.
while / until	The while or until conditional clause must only appear once in any do..loop statement, i.e. either after do or after loop . Each condition is interpreted only the first time it is encountered but is evaluated for every time it encountered in the loop.
exit do	If an exit do clause is encountered inside the loop, the execution of the script will be transferred to the first statement after the loop clause denoting the end of the loop. An exit do clause can be made conditional by the optional use of a when or unless suffix.

Example:

```
// LOAD files file1.csv..file9.csv
Set a=1;
Do while a<10
LOAD * from file$(a).csv;
Let a=a+1;
Loop
```

Exit script

This control statement stops script execution. It may be inserted anywhere in the script.

Syntax:

```
Exit Script [ (when | unless) condition ]
```

Since the **exit script** statement is a control statement and as such is ended with either a semicolon or end-of-line, it must not cross a line boundary.

Arguments:

Argument	Description
condition	A logical expression evaluating to True or False.
when / unless	An exit script statement can be made conditional by the optional use of when or unless clause.

Examples:

```
//Exit script
Exit Script;
```

```
//Exit script when a condition is fulfilled
Exit Script when a=1
```

For..next

The **for..next** control statement is a script iteration construct with a counter. The statements inside the loop enclosed by **for** and **next** will be executed for each value of the counter variable between specified low and high limits.

Syntax:

```
For counter = expr1 to expr2 [ step expr3 ]
[statements]
[exit for [ ( when | unless ) condition ]
[statements]
Next [counter]
```

The expressions *expr1*, *expr2* and *expr3* are only evaluated the first time the loop is entered. The value of the counter variable may be changed by statements inside the loop, but this is not good programming practice.

If an **exit for** clause is encountered inside the loop, the execution of the script will be transferred to the first statement after the **next** clause denoting the end of the loop. An **exit for** clause can be made conditional by the optional use of a **when** or **unless** suffix.



*Since the **for..next** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its three possible clauses (**for..to..step**, **exit for** and **next**) must not cross a line boundary.*

Arguments:

Argument	Description
counter	A variable name. If <i>counter</i> is specified after next it must be the same variable name as the one found after the corresponding for .

Argument	Description
expr1	An expression which determines the first value of the <i>counter</i> variable for which the loop should be executed.
expr2	An expression which determines the value indicating the increment of the <i>counter</i> variable each time the loop has been executed.
expr3	An expression which determines the value indicating the increment of the <i>counter</i> variable each time the loop has been executed.
condition	a logical expression evaluating to True or False.
statements	Any group of one or more Qlik Sense script statements.

Example 1: Loading a sequence of files

```
// LOAD files file1.csv..file9.csv
for a=1 to 9
    LOAD * from file$(a).csv;
next
```

Example 2: Loading a random number of files

In this example, we assume there are data files *x1.csv*, *x3.csv*, *x5.csv*, *x7.csv* and *x9.csv*. Loading is stopped at a random point using the `if rand()<0.5 then` condition.

```
for counter=1 to 9 step 2
    set filename=x$(counter).csv;
    if rand( )<0.5 then
        exit for unless counter=1
    end if
    LOAD a,b from $(filename);
next
```

For each..next

The **for each..next** control statement is a script iteration construct which executes one or several statements for each value in a comma separated list. The statements inside the loop enclosed by **for** and **next** will be executed for each value of the list.

Syntax:

Special syntax makes it possible to generate lists with file and directory names in the current directory.

```
for each var in list
[statements]
[exit for [ ( when | unless ) condition ]
[statements]
next [var]
```

Arguments:

Argument	Description
var	A script variable name which will acquire a new value from list for each loop execution. If var is specified after next it must be the same variable name as the one found after the corresponding for each .

The value of the **var** variable may be changed by statements inside the loop, but this is not good programming practice.

If an **exit for** clause is encountered inside the loop, the execution of the script will be transferred to the first statement after the **next** clause denoting the end of the loop. An **exit for** clause can be made conditional by the optional use of a **when** or **unless** suffix.




Since the **for each..next** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its three possible clauses (**for each**, **exit for** and **next**) must not cross a line boundary.

Syntax:

```
list := item { , item }
item := constant | (expression) | filelist mask | dirlist mask
```

Argument	Description
constant	Any number or string. Note that a string written directly in the script must be enclosed by single quotes. A string without single quotes will be interpreted as a variable, and the value of the variable will be used. Numbers do not need to be enclosed by single quotes.
expression	An arbitrary expression.
mask	A filename or folder name mask which may include any valid filename characters as well as the standard wildcard characters, * and ?. You can use absolute file paths or lib:// paths.
condition	A logical expression evaluating to True or False.
statements	Any group of one or more Qlik Sense script statements.
filelist mask	This syntax produces a comma separated list of all files in the current directory matching the filename mask. <div data-bbox="391 1854 458 1919" data-label="Image"> </div> <i>This argument supports only library connections in standard mode.</i>

Argument	Description
dirlist mask	This syntax produces a comma separated list of all folders in the current folder matching the folder name mask.

 *This argument supports only library connections in standard mode.*

Example 1: Loading a list of files

```
// LOAD the files 1.csv, 3.csv, 7.csv and xyz.csv
for each a in 1,3,7,'xyz'
  LOAD * from file$(a).csv;
next
```

Example 2: Creating a list of files on disk

This example loads a list of all Qlik Sense related files in a folder.

```
sub DoDir (Root)
  for each Ext in 'qvw', 'qva', 'qvo', 'qvs', 'qvc', 'qvf', 'qvd'

    for each File in filelist (Root&'\'*.' &Ext)

      LOAD
        '$(File)' as Name,
        FileSize( '$(File)' ) as Size,
        FileTime( '$(File)' ) as FileTime
      autogenerate 1;

    next File

  next Ext
  for each Dir in dirlist (Root&'\'*' )

    call DoDir (Dir)

  next Dir

end sub

call DoDir ('lib://MyData')
```

If..then..elseif..else..end if

The **if..then** control statement is a script selection construct forcing the script execution to follow different paths depending on one or several logical conditions.

See also: *if* (page 334) (script and chart function)

Syntax:

```
If condition then
  [ statements ]
{ elseif condition then
  [ statements ] }
[ else
  [ statements ] ]
end if
```

Since the **if..then** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its four possible clauses (**if..then**, **elseif..then**, **else** and **end if**) must not cross a line boundary.

Arguments:

Argument	Description
condition	A logical expression which can be evaluated as True or False.
statements	Any group of one or more Qlik Sense script statements.

Example 1:

```
if a=1 then
    LOAD * from abc.csv;
    SQL SELECT e, f, g from tab1;
end if
```

Example 2:

```
if a=1 then; drop table xyz; end if;
```

Example 3:

```
if x>0 then
    LOAD * from pos.csv;
elseif x<0 then
    LOAD * from neg.csv;
else
    LOAD * from zero.txt;
end if
```

Sub..end sub

The **sub..end sub** control statement defines a subroutine which can be called upon from a **call** statement.

Syntax:

```
Sub name [ ( paramlist ) ] statements end sub
```

Arguments are copied into the subroutine and, if the corresponding actual parameter in the **call** statement is a variable name, copied back out again upon exiting the subroutine.

If a subroutine has more formal parameters than actual parameters passed by a **call** statement, the extra parameters will be initialized to NULL and can be used as local variables within the subroutine.

Since the **sub** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its two clauses (**sub** and **end sub**) must not cross a line boundary.

Arguments:

Argument	Description
name	The name of the subroutine.
paramlist	A comma separated list of variable names for the formal parameters of the subroutine. These can be used as any variable inside the subroutine.
statements	Any group of one or more Qlik Sense script statements.

Example 1:

```
Sub INCR (I,J)
I = I + 1
Exit Sub when I < 10
J = J + 1
End Sub
Call INCR (X,Y)
```

Example 2: - parameter transfer

```
Sub ParTrans (A,B,C)
A=A+1
B=B+1
C=C+1
End Sub
A=1
X=1
C=1
Call ParTrans (A, (X+1)*2)
```

The result of the above will be that locally, inside the subroutine, A will be initialized to 1, B will be initialized to 4 and C will be initialized to NULL.

When exiting the subroutine, the global variable A will get 2 as value (copied back from subroutine). The second actual parameter "(X+1)*2" will not be copied back since it is not a variable. Finally, the global variable C will not be affected by the subroutine call.

Switch..case..default..end switch

The **switch** control statement is a script selection construct forcing the script execution to follow different paths, depending on the value of an expression.

Syntax:

```
Switch expression {case valuelist [ statements ]} [default statements] end
switch
```



Since the **switch** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its four possible clauses (**switch**, **case**, **default** and **end switch**) must not cross a line boundary.

Arguments:

Argument	Description
expression	An arbitrary expression.
valuelist	A comma separated list of values with which the value of expression will be compared. Execution of the script will continue with the statements in the first group encountered with a value in valuelist equal to the value in expression. Each value in valuelist may be an arbitrary expression. If no match is found in any case clause, the statements under the default clause, if specified, will be executed.
statements	Any group of one or more Qlik Sense script statements.

Example:

```
Switch I
Case 1
LOAD '$(I): CASE 1' as case autogenerate 1;
Case 2
LOAD '$(I): CASE 2' as case autogenerate 1;
Default
LOAD '$(I): DEFAULT' as case autogenerate 1;
End Switch
```

Script prefixes

Prefixes may be applied to applicable regular statements but never to control statements. The **when** and **unless** prefixes can however be used as suffixes to a few specific control statement clauses.

All script keywords can be typed with any combination of lower case and upper case characters. Field and variable names used in the statements are however case sensitive.

Script prefixes overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Add

The **add** prefix can be added to any **LOAD**, **SELECT** or **map...using** statement in the script. It is only relevant during partial reloads.

```
Add [only] (loadstatement | selectstatement | mapstatement)
```

Buffer

QVD files can be created and maintained automatically via the **buffer** prefix. This prefix can be used on most **LOAD** and **SELECT** statements in script. It indicates that QVD files are used to cache/buffer the result of the statement.

```
Buffer[(option [ , option])] ( loadstatement | selectstatement )  
option::= incremental | stale [after] amount [(days | hours)]
```

Bundle

The **Bundle** prefix is used to include external files, such as image or sound files, or objects connected to a field value, to be stored in the qvf file.

```
Bundle [Info] ( loadstatement | selectstatement)
```

Concatenate

If two tables that are to be concatenated have different sets of fields, concatenation of two tables can still be forced with the **Concatenate** prefix.

```
Concatenate [ (tablename ) ] ( loadstatement | selectstatement )
```

Crosstable

The **crosstable** prefix is used to turn a cross table into a straight table.

```
Crosstable (attribute field name, data field name [ , n ] ) ( loadstatement  
| selectstatement )
```

First

The **First** prefix to a **LOAD** or **SELECT (SQL)** statement is used for loading a set maximum number of records from a data source table.

```
First n( loadstatement | selectstatement )
```

Generic

The unpacking and loading of a generic database can be done with a **generic** prefix.

```
Generic ( loadstatement | selectstatement )
```

Hierarchy

The **hierarchy** prefix is used to transform a hierarchy table to a table that is useful in a Qlik Sense data model. It can be put in front of a **LOAD** or a **SELECT** statement and will use the result of the loading statement as input for a table transformation.

```
Hierarchy (NodeID, ParentID, NodeName, [ParentName], [PathSource],  
[PathName], [PathDelimiter], [Depth])(loadstatement | selectstatement)
```

HierarchBelongsTo

This prefix is used to transform a hierarchy table to a table that is useful in a Qlik Sense data model. It can be put in front of a **LOAD** or a **SELECT** statement and will use the result of the loading statement as input for a table transformation.

```
HierarchyBelongsTo (NodeID, ParentID, NodeName, AncestorID, AncestorName, [DepthDiff]) (loadstatement | selectstatement)
```

Image_Size

This clause is used with the **Info** prefix to resize images from a database management system to fit in the fields.

```
Info [Image_Size (width,height )] ( loadstatement | selectstatement )
```

Info

The **info** prefix is used to link external information, such as a text file, a picture or a video to a field value.

```
Info( loadstatement | selectstatement )
```

Inner

The **join** and **keep** prefixes can be preceded by the prefix **inner**. If used before **join** it specifies that an inner join should be used. The resulting table will thus only contain combinations of field values from the raw data tables where the linking field values are represented in both tables. If used before **keep**, it specifies that both raw data tables should be reduced to their common intersection before being stored in Qlik Sense. .

```
Inner ( Join | Keep) [ (tablename) ](loadstatement |selectstatement )
```

IntervalMatch

The **IntervalMatch** prefix is used to create a table matching discrete numeric values to one or more numeric intervals, and optionally matching the values of one or several additional keys.

```
IntervalMatch (matchfield)(loadstatement | selectstatement )  
IntervalMatch (matchfield,keyfield1 [ , keyfield2, ... keyfield5 ] )  
(loadstatement | selectstatement )
```

Join

The **join** prefix joins the loaded table with an existing named table or the last previously created data table.

```
[Inner | Outer | Left | Right ] Join [ (tablename) ]( loadstatement |  
selectstatement )
```

Keep

The **keep** prefix is similar to the **join** prefix. Just as the **join** prefix, it compares the loaded table with an existing named table or the last previously created data table, but instead of joining the loaded table with an existing table, it has the effect of reducing one or both of the two tables before they are stored in Qlik Sense, based on the intersection of table data. The comparison made is equivalent to a natural join made over all the common fields, i.e. the same way as in a corresponding join. However, the two tables are not joined and

will be kept in Qlik Sense as two separately named tables.

```
(Inner | Left | Right) Keep [(tablename) ]( loadstatement |  
selectstatement )
```

Left

The **Join** and **Keep** prefixes can be preceded by the prefix **left**.

If used before **join** it specifies that a left join should be used. The resulting table will only contain combinations of field values from the raw data tables where the linking field values are represented in the first table. If used before **keep**, it specifies that the second raw data table should be reduced to its common intersection with the first table, before being stored in Qlik Sense.

```
Left ( Join | Keep ) [ (tablename) ](loadstatement |selectstatement )
```

Mapping

The **mapping** prefix is used to create a mapping table that can be used to, for example, replacing field values and field names during script execution.

```
Mapping ( loadstatement | selectstatement )
```

NoConcatenate

The **NoConcatenate** prefix forces two loaded tables with identical field sets to be treated as two separate internal tables, when they would otherwise be automatically concatenated.

```
NoConcatenate ( loadstatement | selectstatement )
```

Outer

The explicit **Join** prefix can be preceded by the prefix **outer** in order to specify an outer join. In an outer join all combinations between the two tables are generated. The resulting table will thus contain combinations of field values from the raw data tables where the linking field values are represented in one or both tables. The **outer** keyword is optional.

```
Outer Join [ (tablename) ](loadstatement |selectstatement )
```

Replace

The **replace** prefix is used to drop the entire Qlik Sense table and replace it with a new table that is loaded or selected.

```
Replace [only] (loadstatement |selectstatement |map...usingstatement)
```

Right

The **Join** and **Keep** prefixes can be preceded by the prefix **right**.

If used before **join** it specifies that a right join should be used. The resulting table will only contain combinations of field values from the raw data tables where the linking field values are represented in the second table. If used before **keep**, it specifies that the first raw data table should be reduced to its common intersection with the second table, before being stored in Qlik Sense.

```
Right (Join | Keep) [(tablename)] (loadstatement | selectstatement )
```

Sample

The **sample** prefix to a **LOAD** or **SELECT** statement is used for loading a random sample of records from the data source.

```
Sample p ( loadstatement | selectstatement )
```

Semantic

Tables containing relations between records can be loaded through a **semantic** prefix. This can for example be self-references within a table, where one record points to another, such as parent, belongs to, or predecessor.

```
Semantic ( loadstatement | selectstatement)
```

Unless

The **unless** prefix and suffix is used for creating a conditional clause which determines whether a statement or exit clause should be evaluated or not. It may be seen as a compact alternative to the full **if..end if** statement.

```
(Unless condition statement | exitstatement Unless condition )
```

When

The **when** prefix and suffix is used for creating a conditional clause which determines whether a statement or exit clause should be executed or not. It may be seen as a compact alternative to the full **if..end if** statement.

```
( When condition statement | exitstatement when condition )
```

Add

The **add** prefix can be added to any **LOAD**, **SELECT** or **map...using** statement in the script. It is only relevant during partial reloads.

Syntax:

```
Add [only] (loadstatement | selectstatement | mapstatement)
```

During a partial reload the Qlik Sense table, for which a table name is generated by the add **LOAD**/add **SELECT** statement (provided such a table exists), will be appended with the result of the add **LOAD**/add **SELECT** statement. No check for duplicates is performed. Therefore, a statement using the **add** prefix will normally include either a **distinct** qualifier or a **where** clause guarding duplicates. The **map...using** statement causes mapping to take place also during partial script execution.

Arguments:

Argument	Description
only	An optional qualifier denoting that the statement should be disregarded during normal (non-partial) reloads.

Examples and results:

Example	Result
Tab1: LOAD Name, Number FROM Persons.csv; Add LOAD Name, Number FROM newPersons.csv;	<p>During normal reload, data is loaded from <i>Persons.csv</i> and stored in the Qlik Sense table Tab1. Data from <i>NewPersons.csv</i> is then concatenated to the same Qlik Sense table.</p> <p>During partial reload, data is loaded from <i>NewPersons.csv</i> and appended to the Qlik Sense table Tab1. No check for duplicates is made.</p>
Tab1: SQL SELECT Name, Number FROM Persons.csv; Add LOAD Name, Number FROM NewPersons.csv where not exists(Name);	<p>A check for duplicates is made by means of looking if Name exists in the previously loaded table data (see the function exists under inter-record functions).</p> <p>During normal reload, data is loaded from <i>Persons.csv</i> and stored in the Qlik Sense table Tab1. Data from <i>NewPersons.csv</i> is then concatenated to the same Qlik Sense table.</p> <p>During partial reload, data is loaded from <i>NewPersons.csv</i> which is appended to the Qlik Sense table Tab1. A check for duplicates is made by means of seeing if Name exists in the previously loaded table data.</p>
Tab1: LOAD Name, Number FROM Persons.csv; Add Only LOAD Name, Number FROM NewPersons.csv where not exists(Name);	<p>During normal reload, data is loaded from <i>Persons.csv</i> and stored in the Qlik Sense table Tab1. The statement loading <i>NewPersons.csv</i> is disregarded.</p> <p>During partial reload, data is loaded from <i>NewPersons.csv</i> which is appended to the Qlik Sense table Tab1. A check for duplicates is made by means of seeing if Name exists in the previously loaded table data.</p>

Buffer

QVD files can be created and maintained automatically via the **buffer** prefix. This prefix can be used on most **LOAD** and **SELECT** statements in script. It indicates that QVD files are used to cache/buffer the result of the statement.

Syntax:

```
Buffer [(option [ , option])] ( loadstatement | selectstatement )
option::= incremental | stale [after] amount [(days | hours)]
```

If no option is used, the QVD buffer created by the first execution of the script will be used indefinitely.

QVD buffers will normally be removed when no longer referenced anywhere throughout a complete script execution in the app that created it or when the app that created it no longer exists.

Arguments:

Argument	Description
incremental	The incremental option enables the ability to read only part of an underlying file. Previous size of the file is stored in the XML header in the QVD file. This is particularly useful with log files. All records loaded at a previous occasion are read from the QVD file whereas the following new records are read from the original source and finally an updated QVD-file is created. Note that the incremental option can only be used with LOAD statements and text files and that incremental load cannot be used where old data is changed or deleted!
stale [after] amount [(days hours)]	amount is a number specifying the time period. Decimals may be used. The unit is assumed to be days if omitted. The stale after option is typically used with DB sources where there is no simple timestamp on the original data. Instead you specify how old the QVD snapshot can be to be used. A stale after clause simply states a time period from the creation time of the QVD buffer after which it will no longer be considered valid. Before that time the QVD buffer will be used as source for data and after that the original data source will be used. The QVD buffer file will then automatically be updated and a new period starts.

Limitations:

Numerous limitations exist, most notable is that there must be either a file **LOAD** or a **SELECT** statement at the core of any complex statement.

Example 1:

```
Buffer SELECT * from MyTable;
```

Example 2:

```
Buffer (stale after 7 days) SELECT * from MyTable;
```

Example 3:

```
Buffer (incremental) LOAD * from MyLog.log;
```

Bundle

The **Bundle** prefix is used to include external files, such as image or sound files, or objects connected to a field value, to be stored in the qvf file.

Syntax:

```
Bundle [Info] ( loadstatement | selectstatement)
```

In order to maintain portability, it is possible to include the external files into the .qvf file itself. To this end use the **Bundle** prefix. The bundled info files are compressed in the process, but will nevertheless take up additional space both in the file and in RAM. Therefore consider both the size and the number of bundled files before going for this solution.

The info may be referenced from the layout as normal info, via the chart info function or as an internal file via the special syntax **qmem:// fieldname / fieldvalue** alternatively **qmem:// fieldname / < index >** where index is the internal index of a field value.

Arguments:

Argument	Description
Info	If a piece of external information, such as an image or a sound file is to be connected to a field value, this is done in a table that is loaded with the Info prefix. The Info prefix may be omitted when Bundle is used.

Example:

```
Bundle Info LOAD * From flagoecd.csv;
Bundle SQL SELECT * from infotable;
```

Concatenate

If two tables that are to be concatenated have different sets of fields, concatenation of two tables can still be forced with the **Concatenate** prefix. This statement forces concatenation with an existing named table or the latest previously created logical table.

Syntax:

```
Concatenate [ (tablename ) ] ( loadstatement | selectstatement )
```

A concatenation is in principle the same as the **SQL UNION** statement, but with two differences:

- The **Concatenate** prefix can be used no matter if the tables have identical field names or not.
- Identical records are not removed with the **Concatenate** prefix.

Arguments:

Argument	Description
tablename	The name of the existing table.

Example:

```
Concatenate LOAD * From file2.csv;
Concatenate SELECT * From table3;
tab1:
LOAD * From file1.csv;
```



```
tab2:
LOAD * From file2.csv;
.. .. ..
Concatenate (tab1) LOAD * From file3.csv;
```

Crosstable

The **crosstable** prefix is used to turn a cross table into a straight table.

Syntax:

```
crosstable (attribute field name, data field name [ , n ] ) ( loadstatement
| selectstatement )
```

Arguments:

Argument	Description
attribute field name	The field that contains the attribute values.
data field name	The field that contains the data values.
n	The number of qualifier fields preceding the table to be transformed to generic form. Default is 1.

A crosstable is a common type of table featuring a matrix of values between two or more orthogonal lists of header data, of which one is used as column headers. A typical example could be to have one column per month. The result of the **crosstable** prefix is that the column headers (for example month names) will be stored in one field, the attribute field, and the column data (month numbers) will be stored in a second field: the data field.

Examples:

```
Crosstable (Month, Sales) LOAD * from ex1.csv;
Crosstable (Month,Sales,2) LOAD * from ex2.csv;
Crosstable (A,B) SELECT * from table3;
```

First

The **First** prefix to a **LOAD** or **SELECT (SQL)** statement is used for loading a set maximum number of records from a data source table.

Syntax:

```
First n ( loadstatement | selectstatement )
```

Arguments:

Argument	Description
n	An arbitrary expression that evaluates to an integer indicating the maximum number of records to be read. <i>n</i> can be enclosed in parentheses, like (<i>n</i>), but this is not required.

Examples:

```
First 10 LOAD * from abc.csv;  
First (1) SQL SELECT * from Orders;
```

Generic

The unpacking and loading of a generic database can be done with a **generic** prefix.

Syntax:

```
Generic( loadstatement | selectstatement )
```

Tables loaded through a **generic** statement are not auto-concatenated.

Examples:

```
Generic LOAD * from abc.csv;  
Generic SQL SELECT * from table1;
```

Hierarchy

The **hierarchy** prefix is used to transform a hierarchy table to a table that is useful in a Qlik Sense data model. It can be put in front of a **LOAD** or a **SELECT** statement and will use the result of the loading statement as input for a table transformation.

Syntax:

```
Hierarchy (NodeID, ParentID, NodeName, [ParentName], [PathSource],  
[PathName], [PathDelimiter], [Depth])(loadstatement | selectstatement)
```

The input table must be an adjacent nodes table. Adjacent nodes tables are tables where each record corresponds to a node and has a field that contains a reference to the parent node. In such a table the node is stored on one record only but the node can still have any number of children. The table may of course contain additional fields describing attributes for the nodes.

The prefix creates an expanded nodes table, which normally has the same number of records as the input table, but in addition each level in the hierarchy is stored in a separate field. The path field can be used in a tree structure.

Usually the input table has exactly one record per node and in such a case the output table will contain the same number of records. However, sometimes there are nodes with multiple parents, i.e. one node is represented by several records in the input table. If so, the output table may have more records than the input table.

All nodes with a parent id not found in the node id column (including nodes with missing parent id) will be considered as roots. Also, only nodes with a connection to a root node - direct or indirect - will be loaded, thus avoiding circular references.

Additional fields containing the name of the parent node, the path of the node and the depth of the node can be created.

Arguments:

Argument	Description
NodeID	The name of the field that contains the node id. This field must exist in the input table.
ParentID	The name of the field that contains the node id of the parent node. This field must exist in the input table.
NodeName	The name of the field that contains the name of the node. This field must exist in the input table.
ParentName	A string used to name the new ParentName field. If omitted, this field will not be created.
ParentSource	The name of the field that contains the name of the node used to build the node path. Optional parameter. If omitted, NodeName will be used.
PathName	A string used to name the new Path field, which contains the path from the root to the node. Optional parameter. If omitted, this field will not be created.
PathDelimiter	A string used as delimiter in the new Path field. Optional parameter. If omitted, '/' will be used.
Depth	A string used to name the new Depth field, which contains the depth of the node in the hierarchy. Optional parameter. If omitted, this field will not be created.

Example:

```
Hierarchy(NodeID, ParentID, NodeName) LOAD
  NodeID,
  ParentID,
  NodeName,
  Attribute
  From data.xls (biff, embedded labels, table is [Sheet1$]);
```

HierarchyBelongsTo

This prefix is used to transform a hierarchy table to a table that is useful in a Qlik Sense data model. It can be put in front of a **LOAD** or a **SELECT** statement and will use the result of the loading statement as input for a table transformation.

Syntax:

```
HierarchyBelongsTo (NodeID, ParentID, NodeName, AncestorID, AncestorName,  
[DepthDiff]) (loadstatement | selectstatement)
```

The input table must be an adjacent nodes table. Adjacent nodes tables are tables where each record corresponds to a node and has a field that contains a reference to the parent node. In such a table the node is stored on one record only but the node can still have any number of children. The table may of course contain additional fields describing attributes for the nodes.

The prefix creates a table containing all ancestor-child relations of the hierarchy. The ancestor fields can then be used to select entire trees in the hierarchy. The output table in most cases contains several records per node.

An additional field containing the depth difference of the nodes can be created.

Arguments:

Argument	Description
NodeID	The name of the field that contains the node id. This field must exist in the input table.
ParentID	The name of the field that contains the node id of the parent node. This field must exist in the input table.
NodeName	The name of the field that contains the name of the node. This field must exist in the input table.
AncestorID	A string used to name the new ancestor id field, which contains the id of the ancestor node.
AncestorName	A string used to name the new ancestor field, which contains the name of the ancestor node.
DepthDiff	A string used to name the new DepthDiff field, which contains the depth of the node in the hierarchy relative the ancestor node. Optional parameter. If omitted, this field will not be created.

Example:

```
HierarchyBelongsTo (NodeID, ParentID, Node, Tree, ParentName) LOAD  
    NodeID,  
    ParentID,  
    NodeName  
    From data.xls (biff, embedded labels, table is [Sheet1$];
```

Image_Size

This clause is used with the **Info** prefix to resize images from a database management system to fit in the fields.

Syntax:

```
Info [Image_Size(width,height )] ( loadstatement | selectstatement )
```

Arguments:

Argument	Description
width	The width of the image specified in pixels.
height	The height of the image specified in pixels.

Example:

```
Info Image_Size(122,122) SQL SELECT ID, Photo From infotable;
```

Info

The **info** prefix is used to link external information, such as a text file, a picture or a video to a field value.

Syntax:

```
Info( loadstatement | selectstatement )
```

If a piece of external information, such as a text file, a picture or a video is to be linked to a field value, this is done in a table that is loaded using an **info** prefix. (In some cases it will be preferable to store the information inside the qvf file, by using the **bundle** prefix. The table must contain two columns only, the first one with the field values that will form the keys to the information, the second one containing the information elements, that is, the file names of the pictures etcetera.

The same applies to, for example, a picture from a database management system. On a binary field, a blob, the info select statement makes an implicit **bundle**, i.e. the binary data will be fetched immediately and stored in the qvf. The binary data must be the second field in a **SELECT** statement.

If a picture needs to be resized, the **image_size** clause can be used.

Example:

```
Info LOAD * from flagoecd.csv;  
Info SQL SELECT * from infotable;  
Info SQL SELECT Key, Picture From infotable;
```

Inner

The **join** and **keep** prefixes can be preceded by the prefix **inner**. If used before **join** it specifies that an inner join should be used. The resulting table will thus only contain combinations of field values from the raw data tables where the linking field values are represented in both tables. If used before **keep**, it specifies that both raw data tables should be reduced to their common intersection before being stored in Qlik Sense.

Syntax:

```
Inner ( Join | Keep ) [ (tablename) ] (loadstatement | selectstatement )
```

Arguments:

Argument	Description
tablename	The named table to be compared to the loaded table.
loadstatement or selectstatement	The LOAD or SELECT statement for the loaded table.

Example 1:

Table1

A	B
1	aa
2	cc
3	ee

Table2

A	C
1	xx
4	yy

QVTable:

```
SQL SELECT * From table1;  
inner join SQL SELECT * From table2;
```

QVTable

A	B	C
1	aa	xx

Example 2:

QVTab1:

```
SQL SELECT * From Table1;
```

QVTab2:

```
inner keep SQL SELECT * From Table2;
```

QVTab1

A	B
1	aa

QVTab2

A	C
1	xx

The two tables in the **keep** example are, of course, associated via A.

IntervalMatch

The **IntervalMatch** prefix is used to create a table matching discrete numeric values to one or more numeric intervals, and optionally matching the values of one or several additional keys.

Syntax:

```
IntervalMatch (matchfield) (loadstatement | selectstatement )
IntervalMatch (matchfield, keyfield1 [ , keyfield2, ... keyfield5 ] )
(loadstatement | selectstatement )
```

The **IntervalMatch** prefix must be placed before a **LOAD** or a **SELECT** statement that loads the intervals. The field containing the discrete data points (Time in the example below) and additional keys must already have been loaded into Qlik Sense before the statement with the **IntervalMatch** prefix. The prefix does not by itself read this field from the database table. The prefix transforms the loaded table of intervals and keys to a table that contains an additional column: the discrete numeric data points. It also expands the number of records so that the new table has one record per possible combination of discrete data point, interval and value of the key field(s).

The intervals may be overlapping and the discrete values will be linked to all matching intervals.

In order to avoid undefined interval limits being disregarded, it may be necessary to allow NULL values to map to other fields that constitute the lower or upper limits to the interval. This can be handled by the **NullAsValue** statement or by an explicit test that replaces NULL values with a numeric value well before or after any of the discrete numeric data points.

Arguments:

Argument	Description
matchfield	The field containing the discrete numeric values to be linked to intervals.
keyfield(s)	Fields that contain the additional attributes that are to be matched in the transformation.
loadstatement or selectstatement	Must result in a table, where the first field contains the lower limit of each interval, the second field contains the upper limit of each interval, and in the case of using key matching, the third and any subsequent fields contain the keyfield(s) present in the IntervalMatch statement. The intervals are always closed, i.e. the end points are included in the interval. Non-numeric limits render the interval to be disregarded (undefined).

Example 1:

In the two tables below, the first one defines the start and end times for the production of different orders. The second one lists a number of discrete events. By means of the **IntervalMatch** prefix it is possible to logically connect the two tables in order to find out e.g. which orders were affected by disturbances and which orders were processed by which shifts.

OrderLog

Start	End	Order
01:00	03:35	A
02:30	07:58	B
03:04	10:27	C
07:23	11:43	D

EventLog

Time	Event	Comment
00:00	0	Start of shift 1
01:18	1	Line stop
02:23	2	Line restart 50%
04:15	3	Line speed 100%
08:00	4	Start of shift 2
11:43	5	End of production

First load the two tables as usual, then link the field *Time* to the time intervals defined by the fields *Start* and *End*:

```
SELECT * from OrderLog;
SELECT * from Eventlog;
IntervalMatch ( Time ) SELECT Start, End from OrderLog;
```

The following table box could now be created in Qlik Sense:

Tablebox

Time	Event	Comment	Order	Start	End
00:00	0	Start of shift 1	-	-	-
01:18	1	Line stop	A	01:00	03:35
02:23	2	Line restart 50%	A	01:00	03:35
04:15	3	Line speed 100%	B	02:30	07:58
04:15	3	Line speed 100%	C	03:04	10:27

08:00	4	Start of shift 2	C	03:04	10:27
08:00	4	Start of shift 2	D	07:23	11:43
11:43	5	End of production	D	07:23	11:43

Example 2: (using keyfield)

```
Inner Join IntervalMatch (Date,Key) LOAD FirstDate, LastDate, Key resident Key;
```

Join

The **join** prefix joins the loaded table with an existing named table or the last previously created data table.

Syntax:

```
[inner | outer | left | right ]Join [ (tablename ) ]( loadstatement |
selectstatement )
```

The join is a natural join made over all the common fields. The join statement may be preceded by one of the prefixes **inner**, **outer**, **left** or **right**.

Arguments:

Argument	Description
tablename	The named table to be compared to the loaded table.
loadstatement or selectstatement	The LOAD or SELECT statement for the loaded table.

Example:

```
Join LOAD * from abc.csv;

Join SELECT * from table1;

tab1:
LOAD * from file1.csv;
tab2:
LOAD * from file2.csv;
... ..
join (tab1) LOAD * from file3.csv;
```

Keep

The **keep** prefix is similar to the **join** prefix. Just as the **join** prefix, it compares the loaded table with an existing named table or the last previously created data table, but instead of joining the loaded table with an existing table, it has the effect of reducing one or both of the two tables before they are stored in Qlik Sense, based on the intersection of table data. The comparison made is equivalent to a natural join made over all the common fields, i.e. the same way as in a corresponding join. However, the two tables are not joined and will be kept in Qlik Sense as two separately named tables.

Syntax:

```
(inner | left | right) keep [(tablename) ] ( loadstatement |  
selectstatement )
```

The **keep** prefix must be preceded by one of the prefixes **inner**, **left** or **right**.

The explicit **join** prefix in Qlik Sense script language performs a full join of the two tables. The result is one table. In many cases such joins will result in very large tables. One of the main features of Qlik Sense is its ability to make associations between multiple tables instead of joining them, which greatly reduces memory usage, increases processing speed and offers enormous flexibility. Explicit joins should therefore generally be avoided in Qlik Sense scripts. The keep functionality was designed to reduce the number of cases where explicit joins needs to be used.

Arguments:

Argument	Description
tablename	The named table to be compared to the loaded table.
loadstatement or selectstatement	The LOAD or SELECT statement for the loaded table.

Example:

```
Inner Keep LOAD * from abc.csv;  
Left Keep SELECT * from table1;  
tab1:  
LOAD * from file1.csv;  
tab2:  
LOAD * from file2.csv;  
.. .. ..  
Left Keep (tab1) LOAD * from file3.csv;
```

Left

The **Join** and **Keep** prefixes can be preceded by the prefix **left**.

If used before **join** it specifies that a left join should be used. The resulting table will only contain combinations of field values from the raw data tables where the linking field values are represented in the first table. If used before **keep**, it specifies that the second raw data table should be reduced to its common intersection with the first table, before being stored in Qlik Sense.

Syntax:

```
Left ( Join | Keep ) [ (tablename) ] (loadstatement | selectstatement)
```

Arguments:

Argument	Description
tablename	The named table to be compared to the loaded table.
loadstatement or selectstatement	The LOAD or SELECT statement for the loaded table.

Example:

Table1	
A	B
1	aa
2	cc
3	ee

Table2	
A	C
1	xx
4	yy

QVTable:

SELECT * From table1;

Left Join Sselect * From table2;

QVTable		
A	B	C
1	aa	xx
2	cc	
3	ee	

QVTab1:

SELECT * From Table1;

QVTab2:

Left Keep SELECT * From Table2;

QVTab1	
A	B
1	aa
2	cc
3	ee

QVTab2	
A	C
1	xx

The two tables in the **keep** example are, of course, associated via A.

```
tab1:
LOAD * From file1.csv;
tab2:
LOAD * From file2.csv;
... ..
Left Keep (tab1) LOAD * From file3.csv;
```

Mapping

The **mapping** prefix is used to create a mapping table that can be used to, for example, replacing field values and field names during script execution.

Syntax:

```
Mapping( loadstatement | selectstatement )
```

The **mapping** prefix can be put in front of a **LOAD** or a **SELECT** statement and will store the result of the loading statement as a mapping table. A mapping table consists of two columns, the first containing comparison values and the second containing the desired mapping values. Mapping tables are stored temporarily in memory and dropped automatically after script execution.

The content of the mapping table can be accessed using e.g. the **Map ... Using** statement, the **Rename Field** statement, the **Applymap()** function or the **Mapsubstring()** function.

Example:

```
Mapping LOAD * from x.csv
Mapping SQL SELECT a, b from map1
map1:
mapping LOAD * inline [
X,Y
US,USA
U.S.,USA
America,USA ];
```

NoConcatenate

The **NoConcatenate** prefix forces two loaded tables with identical field sets to be treated as two separate internal tables, when they would otherwise be automatically concatenated.

Syntax:

```
NoConcatenate( loadstatement | selectstatement )
```

Example:

```
LOAD A,B from file1.csv;
NoConcatenate LOAD A,B from file2.csv;
```

Outer

The explicit **Join** prefix can be preceded by the prefix **outer** in order to specify an outer join. In an outer join all combinations between the two tables are generated. The resulting table will thus contain combinations of field values from the raw data tables where the linking field values are represented in one or both tables. The **outer** keyword is optional.

Syntax:

```
Outer Join [ (tablename) ] (loadstatement |selectstatement )
```

Arguments:

Argument	Description
tablename	The named table to be compared to the loaded table.
loadstatement or selectstatement	The LOAD or SELECT statement for the loaded table.

Example:

Table1	
A	B
1	aa
2	cc
3	ee

Table2	
A	C
1	xx
4	yy

```
SQL SELECT * from table1;
join SQL SELECT * from table2;
    OR
```

```
SQL SELECT * from table1;
outer join SQL SELECT * from table2;
```

Joined table		
A	B	C

1	aa	xx
2	cc	-
3	ee	-
4	-	yy

Replace

The **replace** prefix is used to drop the entire Qlik Sense table and replace it with a new table that is loaded or selected.

Syntax:

```
Replace [only] (loadstatement |selectstatement |map...usingstatement)
```

The **replace** prefix can be added to any **LOAD**, **SELECT** or **map...using** statement in the script.

The **replace LOAD/replace SELECT** statement has the effect of dropping the entire Qlik Sense table, for which a table name is generated by the **replace LOAD/replace SELECT** statement, and replacing it with a new table containing the result of the **replace LOAD/replace SELECT** statement. The effect is the same during partial reload and full reload. The **replace map...using** statement causes mapping to take place also during partial script execution.

Arguments:

Argument	Description
only	An optional qualifier denoting that the statement should be disregarded during normal (non-partial) reloads.

Examples and results:

Example	Result
Tab1: Replace LOAD * from File1.csv;	During both normal and partial reload, the Qlik Sense table Tab1 is initially dropped. Thereafter new data is loaded from File1.csv and stored in Tab1.
Tab1: Replace only LOAD * from File1.csv;	During normal reload, this statement is disregarded. During partial reload, any Qlik Sense table previously named Tab1 is initially dropped. Thereafter new data is loaded from File1.csv and stored in Tab1.
Tab1: LOAD a,b,c from File1.csv; Replace LOAD a,b,c from File2.csv;	During normal reload, the file File1.csv is first read into the Qlik Sense table Tab1, but then immediately dropped and replaced by new data loaded from File2.csv. All data from File1.csv is lost. During partial reload, the entire Qlik Sense table Tab1 is initially dropped. Thereafter it is replaced by new data loaded from File2.csv.

Example	Result
Tab1: LOAD a,b,c from File1.csv; Replace only LOAD a,b,c from File2.csv;	During normal reload, data is loaded from File1.csv and stored in the Qlik Sense table Tab1. File2.csv is disregarded. During partial reload, the entire Qlik Sense table Tab1 is initially dropped. Thereafter it is replaced by new data loaded from File2.csv. All data from File1.csv is lost.

Right

The **Join** and **Keep** prefixes can be preceded by the prefix **right**.

If used before **join** it specifies that a right join should be used. The resulting table will only contain combinations of field values from the raw data tables where the linking field values are represented in the second table. If used before **keep**, it specifies that the first raw data table should be reduced to its common intersection with the second table, before being stored in Qlik Sense.

Syntax:

```
Right (Join | Keep) [(tablename)] (loadstatement | selectstatement )
```

Arguments:

Argument	Description
tablename	The named table to be compared to the loaded table.
loadstatement or selectstatement	The LOAD or SELECT statement for the loaded table.

Examples:

Table1	
A	B
1	aa
2	cc
3	ee

Table2	
A	C
1	xx
4	yy

```
QVTable:
SQL SELECT * from table1;
right join SQL SELECT * from table2;
```

QVTable		
A	B	C
1	aa	xx
4	-	yy

QVTab1:

SQL SELECT * from Table1;

QVTab2:

right keep SQL SELECT * from Table2;

QVTab1	
A	B
1	aa

QVTab2	
A	C
1	xx
4	yy

The two tables in the **keep** example are, of course, associated via A.

tab1:

LOAD * from file1.csv;

tab2:

LOAD * from file2.csv;

.. .. .

right keep (tab1) LOAD * from file3.csv;

Sample

The **sample** prefix to a **LOAD** or **SELECT** statement is used for loading a random sample of records from the data source.

Syntax:

```
Sample p ( loadstatement | selectstatement )
```

Arguments:

Argument	Description
p	An arbitrary expression which evaluates to a number larger than 0 and lower or equal to 1. The number indicates the probability for a given record to be read. All records will be read but only some of them will be loaded into Qlik Sense.

Example:

Sample 0.15 SQL SELECT * from Longtable;
Sample(0.15) LOAD * from Longtab.csv;



The parentheses are allowed but not required.

Semantic

Tables containing relations between records can be loaded through a **semantic** prefix. This can for example be self-references within a table, where one record points to another, such as parent, belongs to, or predecessor.

Syntax:

```
Semantic( loadstatement | selectstatement)
```

The semantic load will create semantic fields that can be displayed in filter panes to be used for navigation in the data.

Tables loaded through a **semantic** statement cannot be concatenated

Example:

```
Semantic LOAD * from abc.csv;  
Semantic SELECT Object1, Relation, Object2, InverseRelation from table1;
```

Unless

The **unless** prefix and suffix is used for creating a conditional clause which determines whether a statement or exit clause should be evaluated or not. It may be seen as a compact alternative to the full **if..end if** statement.

Syntax:

```
(Unless condition statement | exitstatement Unless condition )
```

The **statement** or the **exitstatement** will only be executed if **condition** is evaluated to False.

The **unless** prefix may be used on statements which already have one or several other statements, including additional **when** or **unless** prefixes.

Arguments:

Argument	Description
condition	A logical expression evaluating to True or False.
statement	Any Qlik Sense script statement except control statements.
exitstatement	An exit for , exit do or exit sub clause or an exit script statement.

Examples:

```
exit script unless A=1;
unless A=1 LOAD * from myfile.csv;
unless A=1 when B=2 drop table Tab1;
```

When

The **when** prefix and suffix is used for creating a conditional clause which determines whether a statement or exit clause should be executed or not. It may be seen as a compact alternative to the full **if..end if** statement.

Syntax:

```
(when condition statement | exitstatement when condition )
```

The **statement** or the **exitstatement** will only be executed if condition is evaluated to True.

The **when** prefix may be used on statements which already have one or several other statements, including additional **when** or **unless** prefixes.

Syntax:

Argument	Description
condition	A logical expression evaluating to True or False.
statement	Any Qlik Sense script statement except control statements.
exitstatement	An exit for , exit do or exit sub clause or an exit script statement.

Example 1:

```
exit script when A=1;
```

Example 2:

```
when A=1 LOAD * from myfile.csv;
```

Example 3:

```
when A=1 unless B=2 drop table Tab1;
```

Script regular statements

Regular statements are typically used for manipulating data in one way or another. These statements may be written over any number of lines in the script and must always be terminated by a semicolon, ";".

All script keywords can be typed with any combination of lower case and upper case characters. Field and variable names used in the statements are however case sensitive.

Script regular statements overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Alias

The **alias** statement is used for setting an alias according to which a field will be renamed whenever it occurs in the script that follows.

```
Alias fieldname as aliasname {,fieldname as aliasname}
```

Binary

The **binary** statement is used for loading the data from another Qlik Sense app or QlikView 11.2 or earlier document, including section access data.

```
Binary file  
file ::= [ path ] filename
```

comment

Provides a way of displaying the field comments (metadata) from databases and spreadsheets. Field names not present in the app will be ignored. If multiple occurrences of a field name are found, the last value is used.

```
Comment field *fieldlist using mapname  
Comment field fieldname with comment
```

comment table

Provides a way of displaying the table comments (metadata) from databases or spreadsheets.

```
Comment table tablelist using mapname  
Comment table tablename with comment
```

Connect

The **CONNECT** statement is used to define Qlik Sense access to a general database through the OLE DB/ODBC interface. For ODBC, the data source first needs to be specified using the ODBC administrator.

```
ODBC Connect TO connect-string [ ( access_info ) ]  
OLEDB CONNECT TO connect-string [ ( access_info ) ]  
CUSTOM CONNECT TO connect-string [ ( access_info ) ]  
LIB CONNECT TO connection
```

Direct Query

The **DIRECT QUERY** statement allows you to access tables through an ODBC or OLE DB connection using the Direct Discovery function.

```
Direct Query [path]
```

Directory

The **Directory** statement defines which directory to look in for data files in subsequent **LOAD** statements, until a new **Directory** statement is made.

```
Directory [path]
```

Disconnect

The **Disconnect** statement terminates the current ODBC/OLE DB/Custom connection. This statement is optional.

```
Disconnect
```

drop field

One or several Qlik Sense fields can be dropped from the data model, and thus from memory, at any time during script execution, by means of a **drop field** statement.



*Both **drop field** and **drop fields** are allowed forms with no difference in effect. If no table is specified, the field will be dropped from all tables where it occurs.*

```
Drop field fieldname [ , fieldname2 ...] [from tablename1 [ , tablename2 ...]]
```

```
drop fields fieldname [ , fieldname2 ...] [from tablename1 [ , tablename2 ...]]
```

drop table

One or several Qlik Sense internal tables can be dropped from the data model, and thus from memory, at any time during script execution, by means of a **drop table** statement.



*The forms **drop table** and **drop tables** are both accepted.*

```
Drop table tablename [, tablename2 ...]
```

```
drop tables[ tablename [, tablename2 ...]]
```

Execute

The **Execute** statement is used to run other programs while Qlik Sense is loading data. For example, to make conversions that are necessary.

```
Execute commandline
```

FlushLog

The **FlushLog** statement forces Qlik Sense to write the content of the script buffer to the script log file.

```
FlushLog
```

Force

The **force** statement forces Qlik Sense to interpret field values of subsequent **LOAD** and **SELECT** statements as written with only upper case letters, with only lower case letters, as always capitalized or as

they appear (mixed). This statement makes it possible to associate field values from tables made according to different conventions.

```
Force ( capitalization | case upper | case lower | case mixed )
```

LOAD

The **LOAD** statement loads fields from a file, from data defined in the script, from a previously loaded table, from a web page, from the result of a subsequent **SELECT** statement or by generating data automatically.

```
Load [ distinct ] *fieldlist  
[ ( from file [ format-spec ] |  
from_field fieldsource [format-spec]  
inline data [ format-spec ] |  
resident table-label |  
autogenerate size ) ]  
[ where criterion | while criterion ]  
[ group_by groupbyfieldlist ]  
[ order_by orderbyfieldlist ]
```

Let

The **let** statement is a complement to the **set** statement, used for defining script variables. The **let** statement, in opposition to the **set** statement, evaluates the expression on the right side of the '=' before it is assigned to the variable.

```
Let variablename=expression
```

Map ... using

The **map ... using** statement is used for mapping a certain field value or expression to the values of a specific mapping table. The mapping table is created through the **Mapping** statement.

```
Map *fieldlist Using mapname
```

NullAsNull

The **NullAsNull** statement turns off the conversion of NULL values to string values previously set by a **NullAsValue** statement.

```
NullAsNull *fieldlist
```

NullAsValue

The **NullAsValue** statement specifies for which fields that NULL should be converted to a value.

```
NullAsValue *fieldlist
```

Qualify

The **Qualify** statement is used for switching on the qualification of field names, i.e. field names will get the table name as a prefix.

```
Qualify *fieldlist
```

Rem

The **rem** statement is used for inserting remarks, or comments, into the script, or to temporarily deactivate script statements without removing them.

```
Rem string
```

Rename Field

This script function renames one or more existing Qlik Sense field(s) after they have been loaded.

```
Rename field (using mapname | oldname to newname{ , oldname to newname })
```

```
Rename Fields (using mapname | oldname to newname{ , oldname to newname })
```

Rename Table

This script function renames one or more existing Qlik Sense internal table(s) after they have been loaded.

```
Rename table (using mapname | oldname to newname{ , oldname to newname })
```

```
Rename Tables (using mapname | oldname to newname{ , oldname to newname })
```

Section

With the **section** statement, it is possible to define whether the subsequent **LOAD** and **SELECT** statements should be considered as data or as a definition of the access rights.

```
Section (access | application)
```

Select

The selection of fields from an ODBC data source or OLE DB provider is made through standard SQL **SELECT** statements. However, whether the **SELECT** statements are accepted depends on the ODBC driver or OLE DB provider used.

```
Select [all | distinct | distinctrow | top n [percent] ] *fieldlist  
From tablelist  
[Where criterion ]  
[Group by fieldlist [having criterion ] ]  
[Order by fieldlist [asc | desc] ]  
[ (Inner | Left | Right | Full)Join tablename on fieldref = fieldref ]
```

Set

The **set** statement is used for defining script variables. These can be used for substituting strings, paths, drives, and so on.

```
Set variablename=string
```

Sleep

The **sleep** statement pauses script execution for a specified time.

```
Sleep n
```

SQL

The **SQL** statement allows you to send an arbitrary SQL command through an ODBC or OLE DB connection.

```
SQL sql_command
```

SQLColumns

The **sqlcolumns** statement returns a set of fields describing the columns of an ODBC or OLE DB data source, to which a **connect** has been made.

```
SQLColumns
```

SQLTables

The **sqltables** statement returns a set of fields describing the tables of an ODBC or OLE DB data source, to which a **connect** has been made.

```
SQLTables
```

SQLTypes

The **sqltypes** statement returns a set of fields describing the types of an ODBC or OLE DB data source, to which a **connect** has been made.

```
SQLTypes
```

Star

The string used for representing the set of all the values of a field in the database can be set through the **star** statement. It affects the subsequent **LOAD** and **SELECT** statements.

```
Star is [ string ]
```

Store

This script function creates a QVD or a CSV file.

```
Store [ *fieldlist from] table into filename [ format-spec ];
```

Tag

This script function provides a way of assigning tags to one or more fields. If an attempt to tag a field name not present in the app is made, the tagging will be ignored. If conflicting occurrences of a field or tag name are found, the last value is used.

```
Tag fields fieldlist using mapname
```

```
Tag field fieldname with tagname
```

Trace

The **trace** statement writes a string to the **Script Execution Progress** window and to the script log file, when used. It is very useful for debugging purposes. Using **\$**-expansions of variables that are calculated prior to the **trace** statement, you can customize the message.

```
Trace string
```

Unmap

The **Unmap** statement disables field value mapping specified by a previous **Map ... Using** statement for subsequently loaded fields.

```
Unmap *fieldlist
```

Unqualify

The **Unqualify** statement is used for switching off the qualification of field names that has been previously switched on by the **Qualify** statement.

```
Unqualify *fieldlist
```

Untag

Provides a way of removing tags from one or more fields. If an attempt to untag a Field name not present in the app is made, the untagging will be ignored. If conflicting occurrences of a field or tag name is found, the last value is used.

```
Untag Field fields fieldlist using mapname
```

```
Untag field fieldname with tagname
```

Alias

The **alias** statement is used for setting an alias according to which a field will be renamed whenever it occurs in the script that follows.

Syntax:

```
alias fieldname as aliasname {,fieldname as aliasname}
```

Arguments:

Argument	Description
fieldname	The name of the field in your source data
aliasname	An alias name you want to use instead

Examples and results:

Example	Result
Alias ID_N as NameID;	
Alias A as Name, B as Number, C as Date;	The name changes defined through this statement are used on all subsequent SELECT and LOAD statements. A new alias can be defined for a field name by a new alias statement at any subsequent position in the script.

Binary

The **binary** statement is used for loading the data from another Qlik Sense app or QlikView 11.2 or earlier document, including section access data.

Syntax:

```
binary file
file ::= [ path ] filename
```

Arguments:

Argument	Description
file	The name of the file, including the file extension .qvw or .qvf.
path	<p>The path to the file as a folder data connection.</p> <p>Example: 'lib://Table Files/'</p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none"> absolute <p>Example: c:\data\</p> <ul style="list-style-type: none"> relative to the app containing this script line. <p>Example: data\</p>



*Only one **binary** statement is allowed in the script and it must be the first statement of the script.*

Examples

Binary lib://MyData/customer.qvw;	
Binary customer.qvw;	
Binary c:\qv\customer.qvw;	

Comment field

Provides a way of displaying the field comments (metadata) from databases and spreadsheets. Field names not present in the app will be ignored. If multiple occurrences of a field name are found, the last value is used.

Syntax:

```
comment [fields] *fieldlist using mapname
```

```
comment [field] fieldname with comment
```

The map table used should have two columns, the first containing field names and the second the comments.

Arguments:

Argument	Description
<i>*fieldlist</i>	A comma separated list of the fields to be commented. Using * as field list indicates all fields. The wildcard characters * and ? are allowed in field names. Quoting of field names may be necessary when wildcards are used.
<i>mapname</i>	The name of a mapping table previously read in a mapping LOAD or mapping SELECT statement.
<i>fieldname</i>	The name of the field that should be commented.
<i>comment</i>	The comment that should be added to the field.

Example 1:

```
commentmap:  
mapping LOAD * inline [  
a,b  
Alpha,This field contains text values  
Num,This field contains numeric values  
];  
comment fields using commentmap;
```

Example 2:

```
comment field Alpha with AFieldContainingCharacters;  
comment field Num with '*A field containing numbers';  
comment Gamma with 'Mickey Mouse field';
```

Comment table

Provides a way of displaying the table comments (metadata) from databases or spreadsheets.

Table names not present in the app are ignored. If multiple occurrences of a table name are found, the last value is used. The keyword can be used to read comments from a data source.

Syntax:

```
comment [tables] tablelist using mapname
```

Arguments:

Argument	Description
<i>tablelist</i>	(table{,table})
<i>mapname</i>	The name of a mapping table previously read in a mapping LOAD or mapping SELECT statement.

Syntax:

To set individual comments, the following syntax is used:

```
comment [table] tablename with comment
```

Arguments:

Argument	Description
<i>tablename</i>	The name of the table that should be commented.
<i>comment</i>	The comment that should be added to the table.

Example 1:

```
Commentmap:
mapping LOAD * inline [
a,b
Main,This is the fact table
Currencies, Currency helper table
];
comment tables using commentmap;
```

Example 2:

```
comment table Main with 'Main fact table';
```

Connect

The **CONNECT** statement is used to define Qlik Sense access to a general database through the OLE DB/ODBC interface. For ODBC, the data source first needs to be specified using the ODBC administrator.



This statement supports only folder data connections in standard mode.

Syntax:

```
ODBC CONNECT TO connect-string
OLEDB CONNECT TO connect-string
CUSTOM CONNECT TO connect-string
LIB CONNECT TO connection
```

Arguments:

Argument	Description
connect-string	<p>connect-string ::= datasourceName { ; conn-spec-item }</p> <p>The connection string is the data source name and an optional list of one or more connection specification items. If the data source name contains blanks, or if any connection specification items are listed, the connection string must be enclosed by quotation marks.</p> <p>datasourceName must be a defined ODBC data source or a string that defines an OLE DB provider.</p> <p>conn-spec-item ::= DBQ=database_specifier DriverID=driver_specifier UID=userid PWD=password</p> <p>The possible connection specification items may differ between different databases. For some databases, also other items than the above are possible. For OLE DB, some of the connection specific items are mandatory and not optional.</p>
connection	The name of a data connection stored in the data load editor.

If the **ODBC** is placed before **CONNECT**, the ODBC interface will be used; else, OLE DB will be used.

Using **LIB CONNECT TO** connects to a database using a stored data connection that was created in the data load editor.

Example 1:

```
ODBC CONNECT TO 'Sales
DBQ=C:\Program Files\Access\Samples\Sales.mdb';
```

The data source defined through this statement is used by subsequent **Select (SQL)** statements, until a new **CONNECT** statement is made.

Example 2:

```
LIB CONNECT TO 'MyDataConnection';
```

Connect32

This statement is used the same way as the **CONNECT** statement, but forces a 64-bit system to use a 32-bit ODBC/OLE DB provider. Not applicable for custom connect.

Connect64

This statement is used the same way as the as the **CONNECT** statement, but forces use of a 64-bit provider. Not applicable for custom connect.

Direct Query

The **DIRECT QUERY** statement allows you to access tables through an ODBC or OLE DB connection using the Direct Discovery function.

Syntax:

```
DIRECT QUERY DIMENSION fieldlist [MEASURE fieldlist] [DETAIL fieldlist]
FROM tablelist
[WHERE where_clause]
```

The **DIMENSION**, **MEASURE**, and **DETAIL** keywords can be used in any order.

The **DIMENSION** and **FROM** keyword clauses are required on all **DIRECT QUERY** statements. The **FROM** keyword must appear after the **DIMENSION** keyword.

The fields specified directly after the **DIMENSION** keyword are loaded in memory and can be used to create associations between in-memory and Direct Discovery data.



*The **DIRECT QUERY** statement cannot contain **DISTINCT** or **GROUP BY** clauses.*

Using the **MEASURE** keyword you can define fields that Qlik Sense is aware of on a “meta level”. The actual data of a measure field resides only in the database during the data load process, and is retrieved on an ad hoc basis driven by the chart expressions that are used in a visualization.

Typically, fields with discrete values that will be used as dimensions should be loaded with the **DIMENSION** keyword, whereas numbers that will be used in aggregations only should be selected with the **MEASURE** keyword.

DETAIL fields provide information or details, like comment fields, that a user may want to display in a drill-to-details table box. **DETAIL** fields cannot be used in chart expressions.

By design, the **DIRECT QUERY** statement is data-source neutral for data sources that support SQL. For that reason, the same **DIRECT QUERY** statement can be used for different SQL databases without change. Direct Discovery generates database-appropriate queries as needed.

Native data-source syntax can be used when the user knows the database to be queried and wants to exploit database-specific extensions to SQL. Native data-source syntax is supported:

- As field expressions in **DIMENSION** and **MEASURE** clauses
- As the content of the **WHERE** clause

Examples:

```
DIRECT QUERY
    DIMENSION Dim1, Dim2
    MEASURE
        NATIVE ('X % Y') AS X_MOD_Y
FROM TableName
DIRECT QUERY
    DIMENSION Dim1, Dim2
    MEASURE X, Y
    FROM TableName
    WHERE NATIVE ('EMAIL MATCHES "\*.EDU"')
```



The following terms are used as keywords and so cannot be used as column or field names without being quoted: *and, as, detach, detail, dimension, distinct, from, in, is, like, measure, native, not, or, where*

Arguments:

Argument	Description
fieldlist	A comma-separated list of field specifications, <i>fieldname {, fieldname}</i> . A field specification can be a field name, in which case the same name is used for the database column name and the Qlik Sense field name. Or a field specification can be a "field alias," in which case a database expression or column name is given a Qlik Sense field name.
tablelist	A list of the names of tables or views in the database from which data will be loaded. Typically, it will be views that contain a JOIN performed on the database.
where_ clause	<p>The full syntax of database WHERE clauses is not defined here, but most SQL "relational expressions" are allowed, including the use of function calls, the LIKE operator for strings, IS NULL and IS NOT NULL, and IN. BETWEEN is not included.</p> <p>NOT is a unary operator, as opposed to a modifier on certain keywords.</p> <p>Examples:</p> <pre>WHERE x > 100 AND "Region Code" IN ('south', 'west') WHERE Code IS NOT NULL and Code LIKE '%prospect' WHERE NOT x in (1,2,3)</pre> <p>The last example can not be written as:</p> <pre>WHERE X NOT in (1,2,3)</pre>

Example:

In this example, a database table called TableName, containing fields Dim1, Dim2, Num1, Num2 and Num3, is used. Dim1 and Dim2 will be loaded into the Qlik Sense dataset.

```
DIRECT QUERY DIMENSTION Dim1, Dim2 MEASURE Num1, Num2, Num3 FROM TableName ;
```

Dim1 and Dim2 will be available for use as dimensions. Num1, Num2 and Num3 will be available for aggregations. Dim1 and Dim2 are also available for aggregations. The type of aggregations for which Dim1 and Dim2 can be used depends on their data types. For example, in many cases **DIMENSION** fields contain string data such as names or account numbers. Those fields cannot be summed, but they can be counted: `count(Dim1)`.



DIRECT QUERY statements are written directly in the script editor. To simplify construction of **DIRECT QUERY** statements, you can generate a **SELECT** statement from a data connection, and then edit the generated script to change it into a **DIRECT QUERY** statement. For example, the **SELECT** statement:

```
SQL SELECT
  SalesOrderID,
  RevisionNumber,
  OrderDate,
  SubTotal
  TaxAmt
FROM MyDB.Sales.SalesOrderHeader;
```

could be changed to the following **DIRECT QUERY** statement:

```
DIRECT QUERY
DIMENSION
  SalesOrderID,
  RevisionNumber,

MEASURE
  SubTotal
  TaxAmt
DETAIL
  OrderDate,
FROM MyDB.Sales.SalesOrderHeader;
```

Direct Discovery field lists

A field list is a comma-separated list of field specifications, *fieldname {, fieldname}*. A field specification can be a field name, in which case the same name is used for the database column name and the field name. Or a field specification can be a field alias, in which case a database expression or column name is given a Qlik Sense field name.

Field names can be either simple names or quoted names. A simple name begins with an alphabetic Unicode character and is followed by any combination of alphabetic or numeric characters or underscores. Quoted names begin with a double quotation mark and contain any sequence of characters. If a quoted name contains double quotation marks, those quotation marks are represented using two adjacent double quotation marks.

Qlik Sense field names are case-sensitive. Database field names may or may not be case-sensitive, depending on the database. A Direct Discovery query preserves the case of all field identifiers and aliases. In the following example, the alias "MyState" is used internally to store the data from the database column "STATEID".

```
DIRECT QUERY Dimension STATEID as MyState Measure AMOUNT from SALES_TABLE;
```

This differs from the result of an **SQL Select** statement with an alias. If the alias is not explicitly quoted, the result contains the default case of column returned by the target database. In the following example, the **SQL Select** statement to an Oracle database creates "MYSTATE," with all upper case letters, as the internal Qlik Sense alias even though the alias is specified as mixed case. The **SQL Select** statement uses the column name returned by the database, which in the case of Oracle is all upper case.

```
SQL Select STATEID as MyState, STATENAME from STATE_TABLE;
```

To avoid this behavior, use the **LOAD** statement to specify the alias.

```
Load STATEID as MyState, STATENAME;  
SQL Select STATEID, STATEMENT from STATE_TABLE;
```

In this example, the "STATEID" column is stored internally by Qlik Sense as "MyState".

Most database scalar expressions are allowed as field specifications. Function calls can also be used in field specifications. Expressions can contain constants that are boolean, numeric, or strings contained in single quotation marks (embedded single quotation marks are represented by adjacent single quotation marks).

Examples:

```
DIRECT QUERY DIMENSION SalesOrderID, RevisionNumber MEASURE SubTotal AS "Sub Total" FROM  
Adventureworks.Sales.SalesOrderHeader  
DIRECT QUERY DIMENSION "SalesOrderID" AS "Sales Order ID" MEASURE SubTotal,TaxAmt,(SubTotal-TaxAmt)  
AS "Net Total" FROM Adventureworks.Sales.SalesOrderHeader  
DIRECT QUERY DIMENSION (2*Radius*3.14159) AS Circumference, Molecules/6.02e23 AS Moles MEASURE Num1  
AS numA FROM TableName  
DIRECT QUERY DIMENSION concat(region, 'code') AS region_code MEASURE Num1 AS NumA FROM TableName
```

Direct Discovery does not support using aggregations in **LOAD** statements. If aggregations are used, the results are unpredictable. A **LOAD** statement such as the following should not be used:

```
DIRECT QUERY DIMENSION stateid, SUM(amount*7) AS MultiFirst MEASURE amount FROM sales_table  
The SUM should not be in the LOAD statement.
```

Direct Discovery also does not support Qlik Sense functions in **Direct Query** statements. For example, the following specification for a **DIMENSION** field results in a failure when the "Mth" field is used as a dimension in a visualization:

```
month(ModifiedDate) as Mth
```

Directory

The **Directory** statement defines which directory to look in for data files in subsequent **LOAD** statements, until a new **Directory** statement is made.



This statement has no effect in standard scripting mode.

Syntax:

```
Directory [path]
```


If the **Directory** statement is issued without a **path** or left out, Qlik Sense will look in the Qlik Sense working directory.

Arguments:

Argument	Description
path	<p>A text that can be interpreted as the path to the qvf file.</p> <p>The path is the path to the file, either:</p> <ul style="list-style-type: none">• absolute Example: c:\data\• relative to the Qlik Sense app working directory. Example: data\• URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. Example: http://www.qlik.com

Example:

```
Directory c:\userfiles\data;
```

Disconnect

The **Disconnect** statement terminates the current ODBC/OLE DB/Custom connection. This statement is optional.

Syntax:

```
Disconnect
```

The connection will be automatically terminated when a new **connect** statement is executed or when the script execution is finished.

Example:

```
Disconnect;
```

Drop field

One or several Qlik Sense fields can be dropped from the data model, and thus from memory, at any time during script execution, by means of a **drop field** statement.



Both **drop field** and **drop fields** are allowed forms with no difference in effect. If no table is specified, the field will be dropped from all tables where it occurs.

Syntax:

```
Drop field fieldname [ , fieldname2 ...] [from tablename1 [ , tablename2 ...]]
Drop fields fieldname [ , fieldname2 ...] [from tablename1 [ , tablename2 ...]]
```

Examples:

```
Drop field A;
Drop fields A,B;
Drop field A from X;
Drop fields A,B from X,Y;
```

Drop table

One or several Qlik Sense internal tables can be dropped from the data model, and thus from memory, at any time during script execution, by means of a **drop table** statement.

Syntax:

```
drop table tablename [, tablename2 ...]
drop tables [ tablename [, tablename2 ...]]
```



The forms **drop table** and **drop tables** are both accepted.

The following items will be lost as a result of this:

- The actual table(s).
- All fields which are not part of remaining tables.
- Field values in remaining fields, which came exclusively from the dropped table(s).

Examples and results:

Example	Result
drop table Orders, Salesmen, T456a;	This line results in three tables being dropped from memory.
Tab1: SQL SELECT* from Trans;	As a result only the aggregates remain in the memory. Trans data is discarded.

Example	Result
<pre>LOAD Customer, Sum(sales) resident Tab1 group by Month; drop table Tab1;</pre>	

Execute

The **Execute** statement is used to run other programs while Qlik Sense is loading data. For example, to make conversions that are necessary.



This statement is not supported in standard mode.

Syntax:

```
execute commandline
```

Arguments:

Argument	Description
<i>commandline</i>	A text that can be interpreted by the operating system as a command line. You can refer to an absolute file path or a lib:// folder path.

If you want to use **Execute** the following conditions need to be met:

- You must run in legacy mode (applicable for Qlik Sense and Qlik Sense Desktop).
- You need to set `OverrideScriptSecurity` to 1 in *Settings.ini* (applicable for Qlik Sense). *Settings.ini* is located in `C:\ProgramData\Qlik\Sense\Engine\` and is generally an empty file.

Do the following:

1. Make a copy of *Settings.ini* and open it in a text editor.
2. Insert an empty line.
3. Type `OverrideScriptSecurity=1`.
4. Save the file.
5. Substitute *Settings.ini* with your edited file.



If Qlik Sense is running as a service, some commands may not behave as expected.

Example:

```
Execute C:\Program Files\Office12\Excel.exe;
Execute lib://win\notepad.exe // win is a folder connection referring to c:\windows
```

FlushLog

The **FlushLog** statement forces Qlik Sense to write the content of the script buffer to the script log file.

Syntax:

```
FlushLog
```

The content of the buffer is written to the log file. This command can be useful for debugging purposes, as you will receive data that otherwise may have been lost in a failed script execution.

Example:

```
FlushLog;
```

Force

The **force** statement forces Qlik Sense to interpret field values of subsequent **LOAD** and **SELECT** statements as written with only upper case letters, with only lower case letters, as always capitalized or as they appear (mixed). This statement makes it possible to associate field values from tables made according to different conventions.

Syntax:

```
Force ( capitalization | case upper | case lower | case mixed )
```

If nothing is specified, force case mixed is assumed. The force statement is valid until a new force statement is made.

The **force** statement has no effect in the access section: all field values loaded are case insensitive.

Examples:

```
Force Capitalization;  
Force Case Upper;  
Force Case Lower;  
Force Case Mixed;
```

Load

The **LOAD** statement loads fields from a file, from data defined in the script, from a previously loaded table, from a web page, from the result of a subsequent **SELECT** statement or by generating data automatically.

Syntax:

```
LOAD [ distinct ] *fieldlist  
[( from file [ format-spec ] |  
from_field fieldsource [format-spec]  
inline data [ format-spec ] |  
resident table-label |
```


```

autogenerate size )]
[ where criterion | while criterion ]
[ group_by groupbyfieldlist ]
[ order_by orderbyfieldlist ]

```

Arguments:

Argument	Description
distinct	distinct is a predicate used if only the first of duplicate records should be loaded.
fieldlist	<p><i>*fieldlist ::= (* field {, field})</i></p> <p>A list of the fields to be loaded. Using * as a field list indicates all fields in the table.</p> <p><i>field ::= (fieldref expression) [as aliasname]</i></p> <p>The field definition must always contain a literal, a reference to an existing field, or an expression.</p> <p><i>fieldref ::= (fieldname @fieldnumber @startpos:endpos [I U R B])</i></p> <p><i>fieldname</i> is a text that is identical to a field name in the table. Note that the field name must be enclosed by straight double quotation marks or square brackets if it contains e.g. spaces. Sometimes field names are not explicitly available. Then a different notation is used:</p> <p><i>@fieldnumber</i> represents the field number in a delimited table file. It must be a positive integer preceded by "@". The numbering is always made from 1 and up to the number of fields.</p> <p><i>@startpos:endpos</i> represents the start and end positions of a field in a file with fixed length records. The positions must both be positive integers. The two numbers must be preceded by "@" and separated by a colon. The numbering is always made from 1 and up to the number of positions. If <i>@startpos:endpos</i> is immediately followed by the characters I or U, the bytes read will be interpreted as a binary signed (I) or unsigned (U) integer (Intel byte order). The number of positions read must be 1, 2 or 4. If <i>@startpos:endpos</i> is immediately followed by the character R, the bytes read will be interpreted as a binary real number (IEEE 32-bit or 64 bit floating point). The number of positions read must be 4 or 8. If <i>@startpos:endpos</i> is immediately followed by the character B, the bytes read will be interpreted as a BCD (Binary Coded Decimal) numbers according to the COMP-3 standard. Any number of bytes may be specified.</p> <p><i>expression</i> can be a numeric function or a string function based on one or several other fields in the same table. For further information, see the syntax of expressions.</p> <p>as is used for assigning a new name to the field.</p>
from	<p>from is used if data should be loaded from a file using a folder or a web file data connection.</p> <p><i>file ::= [path] filename</i></p>

Argument	Description
	<p>Example: 'lib://Table Files/'</p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none"> absolute <p>Example: c:\data\</p> <ul style="list-style-type: none"> relative to the Qlik Sense app working directory. <p>Example: data\</p> <ul style="list-style-type: none"> URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. <p>Example: http://www.qlik.com</p> <p>If the path is omitted, Qlik Sense searches for the file in the directory specified by the Directory statement. If there is no Directory statement, Qlik Sense searches in the working directory, <i>C:\Users\{user}\Documents\Qlik\Sense\Apps</i>.</p> <div style="border: 1px solid #ccc; padding: 10px; margin: 10px 0;"> <p> <i>In a Qlik Sense server installation, the working directory is specified in Qlik Sense Repository Service, by default it is C:\ProgramData\Qlik\Sense\Apps. See the Qlik Management Console help for more information.</i></p> </div> <p>The <i>filename</i> may contain the standard DOS wildcard characters (* and ?). This will cause all the matching files in the specified directory to be loaded. <i>format-spec ::= (fspec-item { , fspec-item })</i> The format specification consists of a list of several format specification items, within brackets.</p>
from_field	<p>from_field is used if data should be loaded from a previously loaded field. <i>fieldsource::=(tablename, fieldname)</i></p> <p>The field is the name of the previously loaded <i>tablename</i> and <i>fieldname</i>. <i>format-spec ::= (fspec-item { , fspec-item })</i> The format specification consists of a list of several format specification items, within brackets.</p>
inline	<p>inline is used if data should be typed within the script, and not loaded from a file. <i>data ::= [text]</i></p> <p>Data entered through an inline clause must be enclosed by double quotation marks or by square brackets. The text between these is interpreted in the same way as the</p>

Argument	Description
	<p>content of a file. Hence, where you would insert a new line in a text file, you should also do it in the text of an inline clause, i.e. by pressing the Enter key when typing the script.</p> <p><i>format-spec ::= (fspec-item { , fspec-item })</i></p> <p>The format specification consists of a list of several format specification items, within brackets.</p>
resident	<p>resident is used if data should be loaded from a previously loaded table.</p> <p><i>table label</i> is a label preceding the LOAD or SELECT statement(s) that created the original table. The label should be given with a colon at the end.</p>
autogenerate	<p>autogenerate is used if data should be automatically generated by Qlik Sense.</p> <p><i>size ::= number</i></p> <p><i>Number</i> is an integer indicating the number of records to be generated. The field list must not contain expressions which require data from a database. Only constants and parameter-free functions (such as <i>rand()</i> or <i>recno()</i>), are allowed in the expressions.</p>
where	<p>where is a clause used for stating whether a record should be included in the selection or not. The selection is included if <i>criterion</i> is True.</p> <p><i>criterion</i> is a logical expression.</p>
while	<p>while is a clause used for stating whether a record should be repeatedly read. The same record is read as long as <i>criterion</i> is True. In order to be useful, a while clause must typically include the IterNo() function.</p> <p><i>criterion</i> is a logical expression.</p>
group_by	<p>group by is a clause used for defining over which fields the data should be aggregated (grouped). The aggregation fields should be included in some way in the expressions loaded. No other fields than the aggregation fields may be used outside aggregation functions in the loaded expressions.</p> <p><i>groupbyfieldlist ::= (fieldname { ,fieldname })</i></p>
order_by	<p>order by is a clause used for sorting the records of a resident table before they are processed by the load statement. The resident table can be sorted by one or more fields in ascending or descending order. The sorting is made primarily by numeric value and secondarily by national ASCII value. This clause may only be used when the data source is a resident table.</p> <p>The ordering fields specify which field the resident table is sorted by. The field can be specified by its name or by its number in the resident table (the first field is number 1).</p> <p><i>orderbyfieldlist ::= fieldname [sortorder] { , fieldname [sortorder] }</i></p> <p><i>sortorder</i> is either <i>asc</i> for ascending or <i>desc</i> for descending. If no <i>sortorder</i> is specified, <i>asc</i> is assumed.</p>

Argument	Description
	<i>fieldname</i> , <i>path</i> , <i>filename</i> and <i>aliasname</i> are text strings representing what the respective names imply. Any field in the source table can be used as <i>fieldname</i> . However, fields created through the as clause (<i>aliasname</i>) are out of scope and cannot be used inside the same load statement.

If no source of data is given by means of a **from**, **inline**, **resident**, **from_field** or **autogenerate** clause, data will be loaded from the result of the immediately succeeding **SELECT** or **LOAD** statement. The succeeding statement should not have a prefix.

Examples:

Loading different file formats

```
// LOAD a delimited data file with default options
LOAD * from data1.csv;

// LOAD a delimited data file from a library connection MyData
LOAD * from 'lib://MyData/data1.csv';

// LOAD a delimited file, specifying comma as delimiter and embedded labels
LOAD * from 'c:\userfiles\data1.csv' (ansi, txt, delimiter is ',', embedded labels);

// LOAD a delimited file specifying tab as delimiter and embedded labels
LOAD * from 'c:\userfiles\data2.txt' (ansi, txt, delimiter is '\t', embedded labels);

// LOAD a dif file with embedded headers
LOAD * from file2.dif (ansi, dif, embedded labels);

// LOAD three fields from a fixed record file without headers
LOAD @1:2 as ID, @3:25 as Name, @57:80 as City from data4.fix (ansi, fix, no labels, header is 0,
record is 80);

//LOAD a QVX file, specifying an absolute path
LOAD * from C:\qdssamples\xyz.qvx (qvx);
```

Selecting certain fields, renaming and calculating fields

```
// LOAD only three specific fields
LOAD FirstName, LastName, Number from data1.csv;

// Rename first field as A and second field as B when loading a file without labels
LOAD @1 as A, @2 as B from data3.txt' (ansi, txt, delimiter is '\t', no labels);

// LOAD Name as a concatenation of FirstName, a space, and LastName
LOAD FirstName&' '&LastName as Name from data1.csv;

//LOAD Quantity, Price and Value (the product of Quantity and Price)
LOAD Quantity, Price, Quantity*Price as Value from data1.csv;
```

Selecting certain records

```
// LOAD only unique records, duplicate records will be discarded
LOAD distinct FirstName, LastName, Number from data1.csv;
```



```
// LOAD only records where the field Litres has a value above zero
LOAD * from Consumption.csv where Litres>0;
```

Loading data not on file and autogenerated data

```
// LOAD a table with inline data, fields CatID and Category
LOAD * Inline
[CatID, Category
0,Regular
1,Occasional
2,Permanent];

// LOAD a table with inline data, fields UserID, Password and Access
LOAD * Inline [UserID, Password, Access
A, ABC456, User
B, VIP789, Admin];

// LOAD a table with 10 000 rows
// Field A will contain the number of the read record (1,2,3,4,5...)
// Field B will contain a random number between 0 and 1
LOAD RecNo( ) as A, rand( ) as B autogenerate(10000);
```



The parenthesis after autogenerate is allowed but not required.

Loading data from previously loaded table

```
tab1:
SELECT A,B,C,D from transtable;

// LOAD fields from already loaded table tab1
LOAD A,B,month(C),A*B+D as E resident tab1;

// LOAD fields from already loaded table tab1 but only records where A is larger than B
LOAD A,A+B+C resident tab1 where A>B;

// LOAD fields from already loaded table tab1 ordered by A
LOAD A,B*C as E resident tab1 order by A;

// LOAD fields from already loaded table tab1, ordered by the first field, then the second field
LOAD A,B*C as E resident tab1 order by 1,2;

// LOAD fields from already loaded table tab1 ordered by C descending, then B ascending,
// then first field descending
LOAD A,B*C as E resident tab1 order by C desc, B asc, 1 des
```

Loading data from previously loaded fields

```
// LOAD field types from previously loaded table Characters as A
LOAD A from_field (Characters, Types)
```

Loading data from succeeding table

```
// LOAD A, B and calculated fields C and D from Table1 that is loaded in succeeding statement
LOAD A, B, if(C>0,'positive','negative') as X, weekday(D) as Y;
SELECT A,B,C,D from Table1;
```

Grouping data

```
// LOAD fields grouped (aggregated) by ArtNo
LOAD ArtNo, round(Sum(TransAmount),0.05) as ArtNoTotal from table.csv group by ArtNo;
```

```
// // LOAD fields grouped (aggregated) by week and ArtNo
LOAD Week, ArtNo, round(Avg(TransAmount),0.05) as WeekArtNOAverages from table.csv group by week,
ArtNo;
```

Reading one record repeatedly

In this example we have a input file Grades.csv containing the grades for each student condensed in one field:

```
Student,Grades
Mike,5234
John,3345
Pete,1234
Paul,3352
```

The grades, in a 1-5 scale, represent Math, English, Science and History. We can separate the grades into separate values by reading each record several times with a **while** clause, using the **IterNo()** function as a counter. In each read, the Grade is extracted with the **Mid** function, and the Subject is selected using the **pick** function. The final **while** clause contains the test to check if all grades have been read (four per student in this case), which means next student record should be read.

MyTab:

```
LOAD Student,
mid(Grades,IterNo( ),1) as Grade,
pick(IterNo( ), 'Math', 'English', 'Science', 'History') as Subject from Grades.csv
while IsNum(mid(Grades,IterNo(),1));
```

The result is a table containing this data:

Student	Subject	Grade
John	English	3
John	History	5
John	Math	3
John	Science	4
Mike	English	2
Mike	History	4
Mike	Math	5
Mike	Science	3
Paul	English	3
Paul	History	2
Paul	Math	3
Paul	Science	5
Pete	English	2
Pete	History	4
Pete	Math	1
Pete	Science	3

Format specification items

Each format specification item defines a certain property of the table file:

```
fspec-item ::= [ansi | oem | mac | UTF-8 | Unicode | txt | fix | dif | biff | ooxml | html | xml |
kml | qvd | qvx | delimiter is char | no eof | embedded labels | explicit labels | no labels | table is
[tablename] | header is n | header is line | header is n lines | comment is string | record is n |
record is line | record is n lines | no quotes | msq | filters (filter specifiers)]
```

Character set

Character set is a file specifier for the **LOAD** statement that defines the character set used in the file.

The **ansi**, **oem** and **mac** specifiers were used in QlikView and will still work. However, they will not be generated when creating the **LOAD** statement with Qlik Sense.

Syntax:

```
utf8 | unicode | ansi | oem | mac | codepage is
```

Arguments:

Argument	Description
utf8	UTF-8 character set
unicode	Unicode character set
ansi	Windows, codepage 1252
oem	DOS, OS/2, AS400 and others
mac	Codepage 10000
codepage is	With the codepage specifier, it is possible to use any Windows codepage as <i>N</i> .

Limitations:

Conversion from the **oem** character set is not implemented for MacOS. If nothing is specified, codepage 1252 is assumed under Windows.

Example:

```
LOAD * from a.txt (utf8, txt, delimiter is ',' , embedded labels)
LOAD * from a.txt (unicode, txt, delimiter is ',' , embedded labels)
LOAD * from a.txt (codepage is 10000, txt, delimiter is ',' , no labels)
```

See also:

▢ [Load \(page 68\)](#)

Table format

The table format is a file specifier for the **LOAD** statement that defines the file type. If nothing is specified, a *.txt* file is assumed.

txt In a delimited text file, *.txt*, the columns in the table are separated by some character.

fix In a fixed record length file, *.fix*, each column is exactly a certain number of characters

wide.

- dif** In a *.dif* file, (Data Interchange Format) a special format for defining the table is used.
- biff** Qlik Sense can also interpret data in standard Excel files by means of the *biff* format (Binary Interchange File Format).
- ooxml** Excel 2007 and later versions use the ooxml *.xlsx* format.
- html** If the table is part of an html page or file, html should be used.
- xml** xml (Extensible Markup Language) is a common markup language that is used to represent data structures in a textual format.
- qvd** The format *qvd* is the proprietary QVD files format, exported from a Qlik Sense app.
- qvx** *qvx* is a file/stream format for high performance output to Qlik Sense.

Delimiter

For delimited table files, an arbitrary delimiter can be specified through the **delimiter is** specifier. This specifier is relevant only for delimited *.txt* files.

Syntax:

```
delimiter is char
```

Arguments:

Argument	Description
char	Specifies a single character from the 127 ASCII characters.

Additionally, the following values can be used:

- "\t"** representing a tab sign, with or without quotation marks.
- "\""** representing a backslash (\) character.
- "spaces"** representing all combinations of one or more spaces. Non-printable characters with an ASCII-value below 32, with the exception of CR and LF, will be interpreted as spaces.

If nothing is specified, **delimiter is ','** is assumed.

Example:

```
LOAD * from a.txt (utf8, txt, delimiter is ',' , embedded labels);
```

See also:

- ▢ [Load \(page 68\)](#)

No eof

The **no eof** specifier is used to disregard end-of-file character when loading delimited **.txt** files.

Syntax:

```
no eof
```

If the **no eof** specifier is used, ASCII character 26, which otherwise denotes end-of-file, is disregarded and can be part of a field value.

It is relevant only for delimited **.txt** files.

Example:

```
LOAD * from a.txt (txt, utf8, embedded labels, delimiter is ' ', no eof);
```

See also:

- ▢ [Load \(page 68\)](#)

Labels

Labels is a file specifier for the **LOAD** statement that defines where in a file the field names can be found.

Syntax:

```
embedded labels|explicit labels|no labels
```

The field names can be found in different places of the file. If the first record contains the field names, **embedded labels** should be used. If there are no field names to be found, **no labels** should be used. In *dif* files, a separate header section with explicit field names is sometimes used. In such a case, **explicit labels** should be used. If nothing is specified, **embedded labels** is assumed, also for *dif* files.

Example 1:

```
LOAD * from a.txt (unicode, txt, delimiter is ',', embedded labels
```

Example 2:

```
LOAD * from a.txt (codePage is 1252, txt, delimiter is ',', no labels)
```

See also:

- ▢ [Load \(page 68\)](#)

Header is

Specifies the header size in table files. An arbitrary header length can be specified through the **header is** specifier. A header is a text section not used by Qlik Sense.

Syntax:

```
header is n
header is line
header is n lines
```

The header length can be given in bytes (**header is n**), or in lines (**header is line** or **header is n lines**). **n** must be a positive integer, representing the header length. If not specified, **header is 0** is assumed. The **header is** specifier is only relevant for table files.

Example:

This is an example of a data source table containing a header text line that should not be interpreted as data by Qlik Sense.

```
*Header line
col1,col2
a,B
c,D
```

Using the **header is 1 lines** specifier, the first line will not be loaded as data. In the example, the **embedded labels** specifier tells Qlik Sense to interpret the first non-excluded line as containing field labels.

```
LOAD col1, col2
FROM 'lib://files/header.txt'
(txt, embedded labels, delimiter is ',', msq, header is 1 lines);
```

The result is a table with two fields, Col1 and Col2.

See also:

▫ [Load \(page 68\)](#)

Record is

For fixed record length files, the record length must be specified through the **record is** specifier.

Syntax:

```
Record is n
Record is line
Record is n lines
```

Arguments:

Argument	Description
n	Specifies the record length in bytes.
line	Specifies the record length in one lines.
n lines	Specifies the record length in lines where n is a positive integer representing the record length.

Limitations:

The **record is** specifier is only relevant for **fix** files.

See also:

▢ [Load \(page 68\)](#)

Quotes

Quotes is a file specifier for the **LOAD** statement that defines whether quotes can be used and the precedence between quotes and separators. For text files only.

Syntax:

```
no quotes
msq
```

If the specifier is omitted, standard quoting is used, that is, the quotes " " or ' ' can be used, but only if they are the first and last non blank character of a field value.

Arguments:

Argument	Description
no quotes	Used if quotation marks are not to be accepted in a text file.
msq	Used to specify modern style quoting, allowing multi-line content in fields. Fields containing end-of-line characters must be enclosed within double quotes. One limitation of the msq option is that single double-quote (") characters appearing as first or last character in field content will be interpreted as start or end of multi-line content, which may lead to unpredicted results in the data set loaded. In this case you should use standard quoting instead, omitting the specifier.

XML

This script specifier is used when loading xml files. Valid options for the **XML** specifier are listed in syntax.

Syntax:

```
xmlsax
```

```
xmlsimple  
pattern is path
```

xmlsax and **xmlsimple** are mutually exclusive, only one can be specified when using xml. When using *pattern*, the file will be read from the start of the specified tag to the end of the tag. If *path* contains spaces, the path must be quoted.



*In order to use **xmlsax**, Microsoft's xml parser MSXML 3.0 or higher must be installed on the computer. MSXML is shipped with e.g. Windows XP and MS Internet Explorer 6. It can also be downloaded from the Microsoft home page.*

See also:

▢ [Load \(page 68\)](#)

KML

This script specifier is used when loading KML files to use in a map visualization.

Syntax:

```
kml
```

The KML file can represent either area data (for example, countries or regions) represented by polygons, or point data (for example, cities or places) represented by points in the form [long, lat].

Let

The **let** statement is a complement to the **set** statement, used for defining script variables. The **let** statement, in opposition to the **set** statement, evaluates the expression on the right side of the '=' before it is assigned to the variable.

Syntax:

```
Let variablename=expression
```

The word **let** may be omitted, but the statement then becomes a control statement. Such a statement without the keyword **let** must be contained within a single script row and may be terminated either with a semicolon or end-of-line.

Examples and results:

Example	Result
Set x=3+4; Let y=3+4	\$(x) will be evaluated as '3+4'

Example	Result
<code>z=\$(y)+1;</code>	<code>\$(y)</code> will be evaluated as ' 7 ' <code>\$(z)</code> will be evaluated as ' 8 '
<code>Let T=now();</code>	<code>\$(T)</code> will be given the value of the current time.

Map

The **map ... using** statement is used for mapping a certain field value or expression to the values of a specific mapping table. The mapping table is created through the **Mapping** statement.

Syntax:

```
Map *fieldlist Using mapname
```

The automatic mapping is done for fields loaded after the **Map ... Using** statement until the end of the script or until an **Unmap** statement is encountered.

The mapping is done last in the chain of events leading up to the field being stored in the internal table in Qlik Sense. This means that mapping is not done every time a field name is encountered as part of an expression, but rather when the value is stored under the field name in the internal table. If mapping on the expression level is required, the **Applymap()** function has to be used instead.

Arguments:

Argument	Description
<i>*fieldlist</i>	A comma separated list of the fields that should be mapped from this point in the script. Using * as field list indicates all fields. The wildcard characters * and ? are allowed in field names. Quoting of field names may be necessary when wildcards are used.
<i>mapname</i>	The name of a mapping table previously read in a mapping load or mapping select statement.

Examples and results:

Example	Result
<code>Map Country Using Cmap;</code>	Enables mapping of the field Country using the map Cmap.
<code>Map A, B, C Using X;</code>	Enables mapping of the fields A, B and C using the map X.
<code>Map * Using GenMap;</code>	Enables mapping of all fields using GenMap.

NullAsNull

The **NullAsNull** statement turns off the conversion of NULL values to string values previously set by a **NullAsValue** statement.

Syntax:

```
NullAsNull *fieldlist
```

The **NullAsValue** statement operates as a switch and can be turned on or off several times in the script, using either a **NullAsValue** or a **NullAsNull** statement.

Arguments:

Argument	Description
*fieldlist	A comma separated list of the fields for which NullAsNull should be turned on. Using * as field list indicates all fields. The wildcard characters * and ? are allowed in field names. Quoting of field names may be necessary when wildcards are used.

Example:

```
NullAsNull A,B;  
LOAD A,B from x.csv;
```

NullAsValue

The **NullAsValue** statement specifies for which fields that NULL should be converted to a value.

Syntax:

```
NullAsValue *fieldlist
```

By default, Qlik Sense considers NULL values to be missing or undefined entities. However, certain database contexts imply that NULL values are to be considered as special values rather than simply missing values.

The fact that NULL values are normally not allowed to link to other NULL values can be suspended by means of the **NullAsValue** statement.

The **NullAsValue** statement operates as a switch and will operate on subsequent loading statements. It can be switched off again by means of the **NullAsNull** statement.

Arguments:

Argument	Description
*fieldlist	A comma separated list of the fields for which NullAsValue should be turned on. Using * as field list indicates all fields. The wildcard characters * and ? are allowed in field names. Quoting of field names may be necessary when wildcards are used.

Example:

```
NullAsValue A,B;
Set NullValue = 'NULL';
LOAD A,B from x.csv;
```

Qualify

The **Qualify** statement is used for switching on the qualification of field names, i.e. field names will get the table name as a prefix.

Syntax:

```
Qualify *fieldlist
```

The automatic join between fields with the same name in different tables can be suspended by means of the **qualify** statement, which qualifies the field name with its table name. If qualified, the field name(s) will be renamed when found in a table. The new name will be in the form of *tablename.fieldname*. *Tablename* is equivalent to the label of the current table, or, if no label exists, to the name appearing after **from** in **LOAD** and **SELECT** statements.

The qualification will be made for all fields loaded after the **qualify** statement.

Qualification is always turned off by default at the beginning of script execution. Qualification of a field name can be activated at any time using a **qualify** statement. Qualification can be turned off at any time using an **Unqualify** statement.



*The **qualify** statement should not be used in conjunction with partial reload!*

Arguments:

Argument	Description
*fieldlist	A comma separated list of the fields for which qualification should be turned on. Using * as field list indicates all fields. The wildcard characters * and ? are allowed in field names. Quoting of field names may be necessary when wildcards are used.

Example 1:

```
Qualify B;
LOAD A,B from x.csv;
LOAD A,B from y.csv;
```

The two tables **x.csv** and **y.csv** are associated only through **A**. Three fields will result: A, x.B, y.B.

Example 2:

In an unfamiliar database, it is often useful to start out by making sure that only one or a few fields are associated, as illustrated in this example:

```
qualify *;  
unqualify TransID;  
SQL SELECT * from tab1;  
SQL SELECT * from tab2;  
SQL SELECT * from tab3;
```

Only **TransID** will be used for associations between the tables *tab1*, *tab2* and *tab3*.

Rem

The **rem** statement is used for inserting remarks, or comments, into the script, or to temporarily deactivate script statements without removing them.

Syntax:

```
Rem string
```

Everything between the **rem** and the next semicolon ; is considered to be a comment.

There are two alternative methods available for making comments in the script:

1. It is possible to create a comment anywhere in the script - except between two quotes - by placing the section in question between */** and **/*.
2. When typing *//* in the script, all text that follows to the right on the same row becomes a comment. (Note the exception *//*: that may be used as part of an Internet address.)

Arguments:

Argument	Description
string	An arbitrary text.

Example:

```
Rem ** This is a comment **;  
/* This is also a comment */  
// This is a comment as well
```

Rename field

This script function renames one or more existing Qlik Sense field(s) after they have been loaded.

Either syntax: **rename field** or **rename fields** can be used.

Syntax:

```
Rename Field (using mapname | oldname to newname{ , oldname to newname })  
Rename Fields (using mapname | oldname to newname{ , oldname to newname })
```

Arguments:

Argument	Description
mapname	The name of a previously loaded mapping table containing one or more pairs of old and new field names.
oldname	The old field name.
newname	The new field name.

Limitations:

Two differently named fields cannot be renamed to having the same name. The script will run without errors, but the second field will not be renamed.

Example 1:

```
Rename Field XAZ0007 to Sales;
```

Example 2:

```
FieldMap:  
Mapping SQL SELECT oldnames, newnames from datadictionary;  
Rename Fields using FieldMap;
```

Rename table

This script function renames one or more existing Qlik Sense internal table(s) after they have been loaded.

Either syntax: **rename table** or **rename tables** can be used.

Syntax:

```
Rename Table (using mapname | oldname to newname{ , oldname to newname })  
Rename Tables (using mapname | oldname to newname{ , oldname to newname })
```

Arguments:

Argument	Description
mapname	The name of a previously loaded mapping table containing one or more pairs of old and new table names.
oldname	The old table name.
newname	The new table name.

Limitations:

Two differently named tables cannot be renamed to having the same name. The script will run without errors, but the second table will not be renamed.

Example 1:

```
Tab1:
SELECT * from Trans;
Rename Table Tab1 to Xyz;
```

Example 2:

```
TabMap:
Mapping LOAD oldnames, newnames from tabnames.csv;
Rename Tables using TabMap;
```

Search

The **Search** statement is used for including or excluding fields in the search tool function.

Syntax:

```
Search Include *fieldlist
Search Exclude *fieldlist
```

You can use several Search statements to refine your selection of fields to include. The statements are evaluated from top to bottom.

Arguments:

Argument	Description
*fieldlist	A comma separated list of the fields to include or exclude from searches in the search tool. Using * as field list indicates all fields. The wildcard characters * and ? are allowed in field names. Quoting of field names may be necessary when wildcards are used.

Example:

Search Include *;	Include all fields in searches in the search tool.
Search Exclude *ID;	Exclude all fields ending with ID from searches in the search tool.
Search Include ProductID;	Include the field ProductID in searches in the search tool.

The combined result of these three statements, in this sequence, is that all fields ending with ID except ProductID are excluded from searches in the search tool.

Section

With the **section** statement, it is possible to define whether the subsequent **LOAD** and **SELECT** statements should be considered as data or as a definition of the access rights.

Syntax:

```
Section (access | application)
```

If nothing is specified, **section application** is assumed. The **section** definition is valid until a new **section** statement is made.

Example:

```
Section access;  
Section application;
```

Select

The selection of fields from an ODBC data source or OLE DB provider is made through standard SQL **SELECT** statements. However, whether the **SELECT** statements are accepted depends on the ODBC driver or OLE DB provider used.

Syntax:

```
Select [all | distinct | distinctrow | top n [percent] ] *fieldlist  
From tablelist  
[where criterion ]  
[group by fieldlist [having criterion ] ]  
[order by fieldlist [asc | desc] ]  
[ (Inner | Left | Right | Full) join tablename on fieldref = fieldref ]
```

Furthermore, several **SELECT** statements can sometimes be concatenated into one through the use of a **union** operator:

```
selectstatement Union selectstatement
```

The **SELECT** statement is interpreted by the ODBC driver or OLE DB provider, so deviations from the general SQL syntax might occur depending on the capabilities of the ODBC drivers or OLE DB provider, for example:

- **as** is sometimes not allowed, i.e. *aliasname* must follow immediately after *fieldname*.
- **as** is sometimes compulsory if an *aliasname* is used.
- **distinct**, **as**, **where**, **group by**, **order by**, or **union** is sometimes not supported.
- The ODBC driver sometimes does not accept all the different quotation marks listed above.



*This is not a complete description of the SQL **SELECT** statement! E.g. **SELECT** statements can be nested, several joins can be made in one **SELECT** statement, the number of functions allowed in expressions is sometimes very large, etc.*

Arguments:

Argument	Description
distinct	distinct is a predicate used if duplicate combinations of values in the selected fields only should be loaded once.
distinctrow	distinctrow is a predicate used if duplicate records in the source table only should be loaded once
*fieldlist	<p>*fieldlist ::= (* field) {, field } A list of the fields to be selected. Using * as field list indicates all fields in the table.</p> <p>fieldlist ::= field {, field } A list of one or more fields, separated by commas.</p> <p>field ::= (fieldref expression) [as aliasname] The expression can e.g. be a numeric or string function based on one or several other fields. Some of the operators and functions usually accepted are: +, -, *, /, & (string concatenation), sum(fieldname), count(fieldname), avg(fieldname)(average), month(fieldname), etc. See the documentation of the ODBC driver for more information.</p> <p>fieldref ::= [tablename.] fieldname The tablename and the fieldname are text strings identical to what they imply. They must be enclosed by straight double quotation marks if they contain e.g. spaces. The as clause is used for assigning a new name to the field.</p>
from	<p>tablelist ::= table {, table } The list of tables that the fields are to be selected from.</p> <p>table ::= tablename [[as] aliasname] The tablename may or may not be put within quotes.</p>
where	<p>where is a clause used for stating whether a record should be included in the selection or not.</p> <p>criterion is a logical expression that can sometimes be very complex. Some of the operators accepted are: numeric operators and functions, =, <> or #(not equal), >, >=, <, <=, and, or, not, exists, some, all, in and also new SELECT statements. See the documentation of the ODBC driver or OLE DB provider for more information.</p>
group by	group by is a clause used for aggregating (group) several records into one. Within one group, for a certain field, all the records must either have the same value, or the field can only be used from within an expression, e.g. as a sum or an average. The expression based on one or several fields is defined in the expression of the field symbol.
having	having is a clause used for qualifying groups in a similar manner to how the where clause is used for qualifying records.
order by	order by is a clause used for stating the sort order of the resulting table of the SELECT statement.

Argument	Description
join	join is a qualifier stating if several tables are to be joined together into one. Field names and table names must be put within quotes if they contain blank spaces or letters from the national character sets. When the script is automatically generated by Qlik Sense, the quotation mark used is the one preferred by the ODBC driver or OLE DB provider specified in the data source definition of the data source in the Connect statement.

Example 1:

```
SELECT * FROM `Categories`;
```

Example 2:

```
SELECT `Category ID`, `Category Name` FROM `Categories`;
```

Example 3:

```
SELECT `Order ID`, `Product ID`,  
`Unit Price` * Quantity * (1-Discount) as NetSales  
FROM `Order Details`;
```

Example 4:

```
SELECT `Order Details`.`Order ID`,  
Sum(`Order Details`.`Unit Price` * `Order Details`.Quantity) as `Result`  
FROM `Order Details`, Orders  
where Orders.`Order ID` = `Order Details`.`Order ID`  
group by `Order Details`.`Order ID`;
```

Set

The **set** statement is used for defining script variables. These can be used for substituting strings, paths, drives, and so on.

Syntax:

```
Set variablename=string
```

Example 1:

```
Set FileToUse=Data1.csv;
```

Example 2:

```
Set Constant="My string";
```

Example 3:

```
Set BudgetYear=2012;
```

Sleep

The **sleep** statement pauses script execution for a specified time.

Syntax:

```
Sleep n
```

Arguments:

Argument	Description
n	Stated in milliseconds, where <i>n</i> is a positive integer no larger than 3600000 (i.e. 1 hour). The value may be an expression.

Example 1:

```
Sleep 10000;
```

Example 2:

```
Sleep t*1000;
```

SQL

The **SQL** statement allows you to send an arbitrary SQL command through an ODBC or OLE DB connection.

Syntax:

```
SQL sql_command
```

Sending SQL statements which update the database will return an error if Qlik Sense has opened the ODBC connection in read-only mode.

The syntax:

```
SQL SELECT * from tab1;
```

is allowed, and is the preferred syntax for **SELECT**, for reasons of consistency. The SQL prefix will, however, remain optional for **SELECT** statements.

Arguments:

Argument	Description
<i>sql_command</i>	A valid SQL command.

Example 1:

SQL Leave;

Example 2:

SQL Execute <storedProc>;

SQLColumns

The **sqlcolumns** statement returns a set of fields describing the columns of an ODBC or OLE DB data source, to which a **connect** has been made.

Syntax:

```
SQLcolumns
```

The fields can be combined with the fields generated by the **sqltables** and **sqltypes** commands in order to give a good overview of a given database. The twelve standard fields are:

TABLE_QUALIFIER

TABLE_OWNER

TABLE_NAME

COLUMN_NAME

DATA_TYPE

TYPE_NAME

PRECISION

LENGTH

SCALE

RADIX

NULLABLE

REMARKS

For a detailed description of these fields, see an ODBC reference handbook.

Example:

```
Connect to 'MS Access 7.0 Database; DBQ=C:\Course3\DataSrc\QWT.mbd';  
SQLcolumns;
```



Some ODBC drivers may not support this command. Some ODBC drivers may produce additional fields.

SQLTables

The **sqltables** statement returns a set of fields describing the tables of an ODBC or OLE DB data source, to which a **connect** has been made.

Syntax:

SQLTables

The fields can be combined with the fields generated by the **sqlcolumns** and **sqltypes** commands in order to give a good overview of a given database. The five standard fields are:

TABLE_QUALIFIER
TABLE_OWNER
TABLE_NAME
TABLE_TYPE
REMARKS

For a detailed description of these fields, see an ODBC reference handbook.

Example:

```
Connect to 'MS Access 7.0 Database; DBQ=C:\Course3\DataSrc\QWT.mbd';  
SQLTables;
```



Some ODBC drivers may not support this command. Some ODBC drivers may produce additional fields.

SQLTypes

The **sqltypes** statement returns a set of fields describing the types of an ODBC or OLE DB data source, to which a **connect** has been made.

Syntax:

SQLTypes

The fields can be combined with the fields generated by the **sqlcolumns** and **sqltables** commands in order to give a good overview of a given database. The fifteen standard fields are:

TYPE_NAME
DATA_TYPE
PRECISION

LITERAL_PREFIX
LITERAL_SUFFIX
CREATE_PARAMS
NULLABLE
CASE_SENSITIVE
SEARCHABLE
UNSIGNED_ATTRIBUTE
MONEY
AUTO_INCREMENT
LOCAL_TYPE_NAME
MINIMUM_SCALE
MAXIMUM_SCALE

For a detailed description of these fields, see an ODBC reference handbook.

Example:

```
Connect to 'MS Access 7.0 Database; DBQ=C:\Course3\DataSrc\QWT.mbd';  
SQLTypes;
```



Some ODBC drivers may not support this command. Some ODBC drivers may produce additional fields.

Star

The string used for representing the set of all the values of a field in the database can be set through the **star** statement. It affects the subsequent **LOAD** and **SELECT** statements.

Syntax:

```
Star is [ string ]
```

Arguments:

Argument	Description
string	An arbitrary text. Note that the string must be enclosed by quotation marks if it contains blanks. If nothing is specified, star is ; is assumed, i.e. there is no star symbol available unless explicitly specified. This definition is valid until a new star statement is made.

Examples:

```
Star is *;
Star is %;
Star is;
```

Store

This script function creates a QVD or a CSV file.

Syntax:

```
Store [ *fieldlist from] table into filename [ format-spec ];
```

The statement will create an explicitly named QVD or CSV file. The statement can only export fields from one data table. If fields from several tables are to be exported, an explicit join must be made previously in the script to create the data table that should be exported.

The text values are exported to the CSV file in UTF-8 format. A delimiter can be specified, see **LOAD**. The **store** statement to a CSV file does not support BIFF export.

Arguments:

Argument	Description
<i>*fieldlist::= (* field) { , field }</i>	<p>A list of the fields to be selected. Using * as field list indicates all fields.</p> <p><i>field::= fieldname [aliasname]</i></p> <p><i>fieldname</i> is a text that is identical to a field name in <i>table</i>. (Note that the field name must be enclosed by straight double quotation marks or square brackets if it contains spaces or other non-standard characters.)</p> <p><i>aliasname</i> is an alternate name for the field to be used in the resulting QVD or CSV file.</p>
<i>table</i>	A script label representing an already loaded table to be used as source for data.
<i>filename</i>	<p>The name of the target file as a folder data connection.</p> <p>Example: 'lib://Table Files/'</p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none"> absolute

Argument	Description
	<p>Example: c:\data</p> <ul style="list-style-type: none"> relative to the Qlik Sense app working directory. <p>Example: data</p> <p>If the path is omitted, Qlik Sense stores the file in the directory specified by the Directory statement. If there is no Directory statement, Qlik Sense stores the file in the working directory, <i>C:\Users\{user}\Documents\Qlik\Sense\Apps</i>.</p>
<i>format-spec ::= (txt qvd)</i>	The format specification consists of the text txt for text files, or the text qvd for qvd files. If the format specification is omitted, qvd is assumed.

Example:

```
Store mytable into xyz.qvd (qvd);
Store * from mytable into xyz.qvd;
Store Name, RegNo from mytable into xyz.qvd;
Store Name as a, RegNo as b from mytable into xyz.qvd;
store mytable into myfile.txt (txt);
store * from mytable into 'lib://FolderConnection/myfile.qvd';
```



The two first examples have identical function.

Tag

This script function provides a way of assigning tags to one or more fields. If an attempt to tag a field name not present in the app is made, the tagging will be ignored. If conflicting occurrences of a field or tag name are found, the last value is used.

Syntax:

```
Tag fields fieldlist using mapname
Tag field fieldname with tagname
```

A field tagged with dimension will be displayed at the top of all field selection controls in Qlik Sense except in the **Edit Expression** dialog.

A field tagged with measure will be displayed at the top of all field selection controls in the **Edit Expression** dialog.

Arguments:

Argument	Description
fieldlist	A comma separated list of the fields that should be tagged from this point in the script.
mapname	The name of a mapping table previously loaded in a mapping Load or mapping Select statement.
fieldname	The name of the field that should be tagged.
tagname	The name of the tag that should be applied to the field.

Example 1:

```
tagmap:
mapping LOAD * inline [
a,b
Alpha,MyTag
Num,MyTag
];
tag fields using tagmap;
```

Example 2:

```
tag field Alpha with 'MyTag2';
```

Trace

The **trace** statement writes a string to the **Script Execution Progress** window and to the script log file, when used. It is very useful for debugging purposes. Using $\$$ -expansions of variables that are calculated prior to the **trace** statement, you can customize the message.

Syntax:

```
Trace string
```

Example 1:

```
Trace Main table loaded;
```

Example 2:

```
Let MyMessage = NoOfRows('MainTable') & ' rows in Main Table';
Trace $(MyMessage);
```

Unmap

The **Unmap** statement disables field value mapping specified by a previous **Map ... Using** statement for subsequently loaded fields.

Syntax:

```
Unmap *fieldlist
```

Arguments:

Argument	Description
*fieldlist	a comma separated list of the fields that should no longer be mapped from this point in the script. Using * as field list indicates all fields. The wildcard characters * and ? are allowed in field names. Quoting of field names may be necessary when wildcards are used.

Examples and results:

Example	Result
Unmap Country;	Disables mapping of field Country.
Unmap A, B, C;	Disables mapping of fields A, B and C.
Unmap * ;	Disables mapping of all fields.

Unqualify

The **Unqualify** statement is used for switching off the qualification of field names that has been previously switched on by the **Qualify** statement.

Syntax:

```
Unqualify *fieldlist
```

Arguments:

Argument	Description
*fieldlist	A comma separated list of the fields for which qualification should be turned on. Using * as field list indicates all fields. The wildcard characters * and ? are allowed in field names. Quoting of field names may be necessary when wildcards are used. Refer to the documentation for the Qualify statement for further information.

Example 1:

```
Unqualify *;
```

Example 2:

```
Unqualify TransID;
```

Untag Field

Provides a way of removing tags from one or more fields. If an attempt to untag a Field name not present in the app is made, the untagging will be ignored. If conflicting occurrences of a field or tag name is found, the last value is used.

Syntax:

```
Untag fields fieldlist using mapname
```

```
Untag field fieldname with tagname
```

Arguments:

Argument	Description
fieldlist	A comma separated list of the fields which tags should be removed.
mapname	The name of a mapping table previously loaded in a mapping LOAD or mapping SELECT statement.
fieldname	The name of the field that should be untagged.
tagname	The name of the tag that should be removed from the field.

Example 1:

```
tagmap:  
mapping LOAD * inline [  
a,b  
Alpha,MyTag  
Num,MyTag  
];  
Untag fields using tagmap;
```

Example 2:

```
Untag field Alpha with MyTag2;];
```

2.4 Script variables

A variable in Qlik Sense is a named entity, containing a single data value. A variable typically acquires its value from a **Let**, **Set** or other control statement. The value of a variable can normally be changed by the user at any time.

Variables can contain numeric or alphanumeric data. If the first character of a variable value is an equals sign '=' Qlik Sense will try to evaluate the value as a formula (Qlik Sense expression) and then display or return the result rather than the actual formula text.

When used, the variable is substituted by its value. Variables can be used in the script for dollar sign expansion and in various control statements. This is very useful if the same string is repeated many times in the script, e.g. a path.

Some special system variables will be set by Qlik Sense at the start of the script execution regardless of their previous values.

When defining a script variable, the syntax:

```
set variablename = string
```

or

```
let variable = expression
```

is used. The **Set** command assigns the text to the right of the equal sign to the variable, whereas the **Let** command evaluates the expression.

Variables are case sensitive.

Examples:

```
set HidePrefix = $ ; // the variable will get the character '$' as value.
```

```
Let vToday = Num(Today()); // returns the date serial number of today.
```

Variable calculation

There are several ways to use variables with calculated values in Qlik Sense, and the result depends on how you define it and how you call it in an expression.

In this example we load some inline data:

```
LOAD * INLINE [  
    Dim, Sales  
    A, 150  
    A, 200  
    B, 240  
    B, 230  
    C, 410  
    C, 330  
];
```

Let's define two variables:

```
Let vSales = 'Sum(Sales)' ;  
Let vSales2 = '=Sum(Sales)' ;
```

In the second variable, we add an equal sign before the expression. This will cause the variable to be calculated before it is expanded and the expression is evaluated.

If you use the vSales variable as it is, for example in a measure, the result will be the string Sum(Sales), that is, no calculation is performed.

If you add a dollar-sign expansion and call \$(vSales) in the expression, the variable is expanded, and the sum of Sales is displayed.

Finally, if you call \$(vSales2), the variable will be calculated before it is expanded. This means that the result displayed is the total sum of Sales. The difference between using=\$(vSales) and=\$(vSales2) as measure expressions is seen in this chart showing the results:

Dim	\$(vSales)	\$(vSales2)
A	350	1560
B	470	1560
C	740	1560

As you can see, \$(vSales) results in the partial sum for a dimension value, while \$(vSales2) results in the total sum.

The following script variables are available:

Error variables	<i>page 116</i>
Number interpretation variables	<i>page 106</i>
System variables	<i>page 100</i>
Value handling variables	<i>page 104</i>

System variables

System variables, some of which are system-defined, provide information about the system and the Qlik Sense app.

System variables overview

Some of the functions are described further after the overview. For those functions, you can click the function name in the syntax to immediately access the details for that specific function.

Floppy

Returns the drive letter of the first floppy drive found, normally a:. This is a system-defined variable.

Floppy



This variable is not supported in standard mode.

CD

Returns the drive letter of the first CD-ROM drive found. If no CD-ROM is found, then c: is returned. This is a system-defined variable.

CD



This variable is not supported in standard mode.

Include

The **include** variable specifies a file that contains text that should be included in the script. The entire script can thus be put in a file. This is a user-defined variable.

```
$(Include =filename)
```

HidePrefix

All field names beginning with this text string will be hidden in the same manner as the system fields. This is a user-defined variable.

HidePrefix

HideSuffix

All field names ending with this text string will be hidden in the same manner as the system fields. This is a user-defined variable.

HideSuffix

QvPath

Returns the browse string to the Qlik Sense executable. This is a system-defined variable.

QvPath



This variable is not supported in standard mode.

QvRoot

Returns the root directory of the Qlik Sense executable. This is a system-defined variable.

QvRoot



This variable is not supported in standard mode.

QvWorkPath

Returns the browse string to the current Qlik Sense app. This is a system-defined variable.

QvWorkPath



This variable is not supported in standard mode.

QvWorkRoot

Returns the root directory of the current Qlik Sense app. This is a system-defined variable.

QvWorkRoot



This variable is not supported in standard mode.

StripComments

If this variable is set to 0, stripping of `/*..*/` and `//` comments in the script will be inhibited. If this variable is not defined, stripping of comments will always be performed.

StripComments

Verbatim

Normally all field values are automatically stripped of leading and trailing blanks (ASCII 32) before being

loaded into the Qlik Sense database. Setting this variable to 1 suspends the stripping of blanks. Tab (ASCII 32) and hard space (ANSI 160) are never stripped.

Verbatim

OpenUrlTimeout

This variable defines the timeout in seconds that Qlik Sense should respect when getting data from URL sources (e.g. HTML pages). If omitted, the timeout is about 20 minutes.

OpenUrlTimeout

WinPath

Returns the browse string to Windows. This is a system-defined variable.

WinPath



This variable is not supported in standard mode.

WinRoot

Returns the root directory of Windows. This is a system-defined variable.

WinRoot



This variable is not supported in standard mode.

CollationLocale

Specifies which locale to use for sort order and search matching. The value is the culture name of a locale, for example 'en-US'. This is a system-defined variable.

CollationLocale

HidePrefix

All field names beginning with this text string will be hidden in the same manner as the system fields. This is a user-defined variable.

Syntax:

HidePrefix

Example:

```
set HidePrefix='_' ;
```

If this statement is used, the field names beginning with an underscore will not be shown in the field name lists when the system fields are hidden.

HideSuffix

All field names ending with this text string will be hidden in the same manner as the system fields. This is a user-defined variable.

Syntax:

```
HideSuffix
```

Example:

```
set HideSuffix='%';
```

If this statement is used, the field names ending with a percentage sign will not be shown in the field name lists when the system fields are hidden.

Include

The **include** variable specifies a file that contains text that should be included in the script. The entire script can thus be put in a file. This is a user-defined variable.



This variable supports only folder data connections in standard mode.

Syntax:

```
$(Include =filename)
```

Example:

```
$(Include=abc.txt);  
$(Include=lib:\\MyDataFiles\\abc.txt);
```

If you don't specify a path, the filename will be relative to the Qlik Sense app working directory. You can also specify an absolute file path, or a path to a lib:\\ folder connection.



*The construction **set Include =filename** is not applicable.*

OpenUrlTimeout

This variable defines the timeout in seconds that Qlik Sense should respect when getting data from URL sources (e.g. HTML pages). If omitted, the timeout is about 20 minutes.

Syntax:

```
OpenUrlTimeout
```

Example:

```
set OpenUrlTimeout=10
```

StripComments

If this variable is set to 0, stripping of `/*..*/` and `//` comments in the script will be inhibited. If this variable is not defined, stripping of comments will always be performed.

Syntax:

StripComments

Certain database drivers use `/*..*/` as optimization hints in **SELECT** statements. If this is the case, the comments should not be stripped before sending the **SELECT** statement to the database driver.



It is recommended that this variable be reset to 1 immediately after the statement(s) where it is needed.

Example:

```
set StripComments=0;
SQL SELECT * /* <optimization directive> */ FROM Table ;
set StripComments=1;
```

Verbatim

Normally all field values are automatically stripped of leading and trailing blanks (ASCII 32) before being loaded into the Qlik Sense database. Setting this variable to 1 suspends the stripping of blanks. Tab (ASCII 32) and hard space (ANSI 160) are never stripped.

Syntax:

Verbatim

Example:

```
set Verbatim = 1;
```

Value handling variables

This section describes variables that are used for handling NULL and other values.

Value handling variables overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

NullDisplay

The defined symbol will substitute all NULL values from ODBC on the lowest level of data. This is a user-

defined variable.

NullDisplay

NullInterpret

The defined symbol will be interpreted as NULL when it occurs in a text file, Excel file or an inline statement. This is a user-defined variable.

NullInterpret

NullValue

If the **NullAsValue** statement is used, the defined symbol will substitute all NULL values in the **NullAsValue** specified fields with the specified string.

NullValue

OtherSymbol

Defines a symbol to be treated as 'all other values' before a **LOAD/SELECT** statement. This is a user-defined variable.

OtherSymbol

NullDisplay

The defined symbol will substitute all NULL values from ODBC on the lowest level of data. This is a user-defined variable.

Syntax:

```
NullDisplay
```

Example:

```
set NullDisplay='<NULL>';
```

NullInterpret

The defined symbol will be interpreted as NULL when it occurs in a text file, Excel file or an inline statement. This is a user-defined variable.

Syntax:

```
NullInterpret
```

Examples:

```
set NullInterpret=' ';  
set NullInterpret =;
```

will NOT return NULL values for blank values in Excel (but it will for a csv text file)

```
set NullInterpret ='';
```

will return NULL values for blank values in Excel (but will NOT for a csv text files)

NullValue

If the **NullAsValue** statement is used, the defined symbol will substitute all NULL values in the **NullAsValue** specified fields with the specified string.

Syntax:

```
NullValue
```

Example:

```
NullAsValue Field1, Field2;  
set NullValue='<NULL>';
```

OtherSymbol

Defines a symbol to be treated as 'all other values' before a **LOAD/SELECT** statement. This is a user-defined variable.

Syntax:

```
OtherSymbol
```

Example:

```
set otherSymbol='+';  
LOAD * inline  
[X, Y  
a, a  
b, b];  
LOAD * inline  
[X, Z  
a, a  
+, c];
```

The field value Y='b' will now link to Z='c' through the other symbol.

Number interpretation variables

The following variables are system defined, i.e. they are automatically generated according to the current regional settings of the operating system when a new app is created. The number interpretation variables are included at the top of the script of the new Qlik Sense app and may substitute operating system defaults for certain number formatting settings at the time of the script execution. They may be deleted, edited or duplicated freely.

Number interpretation variables overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Currency formatting

MoneyDecimalSep

The decimal separator defined replaces the decimal symbol for currency of the operating system (regional settings).

```
MoneyDecimalSep
```

MoneyFormat

The symbol defined replaces the currency symbol of the operating system (regional settings).

```
MoneyFormat
```

MoneyThousandSep

The thousands separator defined replaces the digit grouping symbol for currency of the operating system (regional settings).

```
MoneyThousandSep
```

Number formatting

DecimalSep

The decimal separator defined replaces the decimal symbol of the operating system (regional settings).

```
DecimalSep
```

ThousandSep

The thousands separator defined replaces the digit grouping symbol of the operating system (regional settings).

```
ThousandSep
```

Time formatting

DateFormat

The format defined replaces the date format of the operating system (regional settings).

```
DateFormat
```

TimeFormat

The format defined replaces the time format of the operating system (regional settings).

```
TimeFormat
```

TimestampFormat

The format defined replaces the date and time formats of the operating system (regional settings).

```
TimestampFormat
```

MonthNames

The format defined replaces the month names convention of the operating system (regional settings).

MonthNames

LongMonthNames

The format defined replaces the long month names convention of the operating system (regional settings).

LongMonthNames

DayNames

The format defined replaces the weekday names convention of the operating system (regional settings).

DayNames

LongDayNames

The format defined replaces the long weekday names convention of the operating system (regional settings).

LongDayNames

FirstWeekDay

The integer defines which day to use as the first day of the week.

FirstWeekDay

BrokenWeeks

The setting defines if weeks are broken or not.

BrokenWeeks

ReferenceDay

The setting defines which day in January to set as reference day to define week 1.

ReferenceDay

FirstMonthOfYear

The setting defines which month to use as first month of the year, which can be used to define financial years that use a monthly offset, for example starting April 1.

Valid settings are 1 (January) to 12 (December). Default setting is 1.

Syntax:

FirstMonthOfYear

Example:

```
Set FirstMonthOfYear=4; //Sets the year to start in April
```

BrokenWeeks

The setting defines if weeks are broken or not.

Syntax:

BrokenWeeks

By default, Qlik Sense functions use unbroken weeks. This means that:

- In some years, week 1 starts in December, and in other years, week 52 or 53 continues into January.
- Week 1 always has at least 4 days in January.

The alternative is to use broken weeks.

- Week 52 or 53 do not continue into January.
- Week 1 starts on January 1 and is, in most cases, not a full week.

The following values can be used:

- 0 (=use unbroken weeks)
- 1 (= use broken weeks)

Examples:

```
Set BrokenWeeks=0; //(use unbroken weeks)
Set BrokenWeeks=1; //(use broken weeks)
```

DateFormat

The format defined replaces the date format of the operating system (regional settings).

Syntax:

```
DateFormat
```

Examples:

```
Set DateFormat='M/D/YY'; //(US format)
Set DateFormat='DD/MM/YY'; //(UK date format)
Set DateFormat='YYYY-MM-DD'; //(ISO date format)
```

DayNames

The format defined replaces the weekday names convention of the operating system (regional settings).

Syntax:

```
DayNames
```

Example:

```
Set DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
```

DecimalSep

The decimal separator defined replaces the decimal symbol of the operating system (regional settings).

Syntax:

```
DecimalSep
```

Examples:

```
Set DecimalSep='.';
Set DecimalSep=',';
```

FirstWeekDay

The integer defines which day to use as the first day of the week.

Syntax:

FirstWeekDay

By default, Qlik Sense functions use Monday as the first day of the week. The following values can be used:

- 0 (= Monday)
- 1 (= Tuesday)
- 2 (= Wednesday)
- 3 (= Thursday)
- 4 (= Friday)
- 5 (= Saturday)
- 6 (= Sunday)

Examples:

```
Set FirstWeekDay=6; //(set Sunday as the first day of the week)
```

LongDayNames

The format defined replaces the long weekday names convention of the operating system (regional settings).

Syntax:

LongDayNames

Example:

```
Set LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday';
```

LongMonthNames

The format defined replaces the long month names convention of the operating system (regional settings).

Syntax:

LongMonthNames

Example:

```
Set LongMonthNames='January;February;March;April;May;June - -
```

MoneyDecimalSep

The decimal separator defined replaces the decimal symbol for currency of the operating system (regional settings).

Syntax:

```
MoneyDecimalSep
```

Example:

```
Set MoneyDecimalSep='.';
```

MoneyFormat

The symbol defined replaces the currency symbol of the operating system (regional settings).

Syntax:

```
MoneyFormat
```

Example:

```
Set MoneyFormat='$ #,##0.00; ($ #,##0.00)';
```

MoneyThousandSep

The thousands separator defined replaces the digit grouping symbol for currency of the operating system (regional settings).

Syntax:

```
MoneyThousandSep
```

Example:

```
Set MoneyThousandSep=',';
```

MonthNames

The format defined replaces the month names convention of the operating system (regional settings).

Syntax:

```
MonthNames
```

Example:

```
Set MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';
```

ReferenceDay

The setting defines which day in January to set as reference day to define week 1.

Syntax:

ReferenceDay

By default, Qlik Sense functions use 4 as the reference day. This means that week 1 must contain January 4, or put differently, that week 1 must always have at least 4 days in January.

The following values can be used to set a different reference day:

- 1 (= January 1)
- 2 (= January 2)
- 3 (= January 3)
- 4 (= January 4)
- 5 (= January 5)
- 6 (= January 6)
- 7 (= January 7)

Examples:

```
Set ReferenceDay=3; //(set January 3 as the reference day)
```

ThousandSep

The thousands separator defined replaces the digit grouping symbol of the operating system (regional settings).

Syntax:

ThousandSep

Examples:

```
Set ThousandSep=','; //(for example, seven billion must be specified as: 7,000,000,000)
Set ThousandSep=' ';
```

TimeFormat

The format defined replaces the time format of the operating system (regional settings).

Syntax:

TimeFormat

Example:

```
Set TimeFormat='hh:mm:ss';
```

TimestampFormat

The format defined replaces the date and time formats of the operating system (regional settings).

Syntax:

```
TimestampFormat
```

Example:

```
Set TimestampFormat='M/D/YY hh:mm:ss[.fff]';
```

Direct Discovery variables

Direct Discovery system variables

DirectCacheSeconds

You can set a caching limit to the Direct Discovery query results for visualizations. Once this time limit is reached, Qlik Sense clears the cache when new Direct Discovery queries are made. Qlik Sense queries the source data for the selections and creates the cache again for the designated time limit. The result for each combination of selections is cached independently. That is, the cache is refreshed for each selection independently, so one selection refreshes the cache only for the fields selected, and a second selection refreshes cache for its relevant fields. If the second selection includes fields that were refreshed in the first selection, they are not updated in cache again if the caching limit has not been reached.

The Direct Discovery cache does not apply to **Table** visualizations. Table selections query the data source every time.

The limit value must be set in seconds. The default cache limit is 1800 seconds (30 minutes).

The value used for **DirectCacheSeconds** is the value set at the time the **DIRECT QUERY** statement is executed. The value cannot be changed at runtime.

Example:

```
SET DirectCacheSeconds=1800
```

DirectConnectionMax

You can do asynchronous, parallel calls to the database by using the connection pooling capability. The load script syntax to set up the pooling capability is as follows:

```
SET DirectConnectionMax=10
```

The numeric setting specifies the maximum number of database connections the Direct Discovery code should use while updating a sheet. The default setting is 1.



This variable should be used with caution. Setting it to greater than 1 is known to cause problems when connecting to Microsoft SQL Server.

DirectUnicodeStrings

Direct Discovery can support the selection of extended Unicode data by using the SQL standard format for

extended character string literals (N'<extended string>') as required by some databases (notably SQL Server). The use of this syntax can be enabled for Direct Discovery with the script variable **DirectUnicodeStrings**.

Setting this variable to 'true' will enable the use of the ANSI standard wide character marker "N" in front of the string literals. Not all databases support this standard. The default setting is 'false'.

DirectDistinctSupport

When a **DIMENSION** field value is selected in a Qlik Sense object, a query is generated for the source database. When the query requires grouping, Direct Discovery uses the **DISTINCT** keyword to select only unique values. Some databases, however, require the **GROUP BY** keyword. Set **DirectDistinctSupport** to "false" to generate **GROUP BY** instead of **DISTINCT** in queries for unique values.

```
SET DirectDistinctSupport=false
```

If **DirectDistinctSupport** is set to true, then **DISTINCT** is used. If it is not set, the default behavior is to use **DISTINCT**.

Teradata query banding variables

Teradata query banding is a function that enables enterprise applications to collaborate with the underlying Teradata database in order to provide for better accounting, prioritization, and workload management. Using query banding you can wrap metadata, such as user credentials, around a query.

Two variables are available, both are strings that are evaluated and sent to the database.

SQLSessionPrefix

This string is sent when a connection to the database is created.

```
SET SQLSessionPrefix = 'SET QUERY_BAND = ' & Chr(39) & 'who=' & OSuser() & ';' & Chr(39) & ' FOR SESSION;';
```

If **OSuser()** for example returns *WA\sbt*, this will be evaluated to `SET QUERY_BAND = 'who=WA\sbt;' FOR SESSION;`, which is sent to the database when the connection is created.

SQLQueryPrefix

This string is sent for each single query.

```
SET SQLSessionPrefix = 'SET QUERY_BAND = ' & Chr(39) & 'who=' & OSuser() & ';' & Chr(39) & ' FOR TRANSACTION;';
```

Direct Discovery character variables

DirectFieldColumnDelimiter

You can set the character used as the field delimiter in **Direct Query** statements for databases that require a character other than comma as the field delimiter. The specified character must be surrounded by single quotation marks in the **SET** statement.

```
SET DirectFieldColumnDelimiter= '|'
```

DirectStringQuoteChar

You can specify a character to use to quote strings in a generated query. The default is a single quotation

mark. The specified character must be surrounded by single quotation marks in the **SET** statement.

```
SET DirectStringQuoteChar= '''
```

DirectIdentifierQuoteStyle

You can specify that non-ANSI quoting of identifiers be used in generated queries. At this time, the only non-ANSI quoting available is GoogleBQ. The default is ANSI. Uppercase, lowercase, and mixed case can be used (ANSI, ansi, Ansi).

```
SET DirectIdentifierQuoteStyle="GoogleBQ"
```

For example, ANSI quoting is used in the following **SELECT** statement:

```
SELECT [Quarter] FROM [qvTest].[sales] GROUP BY [Quarter]
```

When **DirectIdentifierQuoteStyle** is set to "GoogleBQ", the **SELECT** statement would use quoting as follows:

```
SELECT [Quarter] FROM [qvTest.sales] GROUP BY [Quarter]
```

DirectIdentifierQuoteChar

You can specify a character to control the quoting of identifiers in a generated query. This can be set to either one character (such as a double quotation mark) or two (such as a pair of square brackets). The default is a double quotation mark.

```
SET DirectIdentifierQuoteChar='YYYY-MM-DD'
```

DirectTableBoxListThreshold

When Direct Discovery fields are used in a **Table** visualization, a threshold is set to limit the number of rows displayed. The default threshold is 1000 records. The default threshold setting can be changed by setting the **DirectTableBoxListThreshold** variable in the load script. For example:

```
SET DirectTableBoxListThreshold=5000
```

The threshold setting applies only to **Table** visualizations that contain Direct Discovery fields. **Table** visualizations that contain only in-memory fields are not limited by the **DirectTableBoxListThreshold** setting.

No fields are displayed in the **Table** visualization until the selection has fewer records than the threshold limit.

Direct Discovery number interpretation variables

DirectMoneyDecimalSep

The decimal separator defined replaces the decimal symbol for currency in the SQL statement generated to load data using Direct Discovery. This character must match the character used in **DirectMoneyFormat**.

Default value is '.'.

Example:

```
Set DirectMoneyDecimalSep='.';
```

DirectMoneyFormat

The symbol defined replaces the currency format in the SQL statement generated to load data using Direct Discovery. The currency symbol for the thousands separator should not be included.

Default value is '#.0000'

Example:

```
Set DirectMoneyFormat='#.0000';
```

DirectTimeFormat

The time format defined replaces the time format in the SQL statement generated to load data using Direct Discovery.

Example:

```
Set DirectTimeFormat='hh:mm:ss';
```

DirectDateFormat

The date format defined replaces the date format in the SQL statement generated to load data using Direct Discovery.

Example:

```
Set DirectDateFormat='MM/DD/YYYY';
```

DirectTimeStampFormat

The format defined replaces the date and time format in the SQL statement generated in the SQL statement generated to load data using Direct Discovery.

Example:

```
Set DirectTimestampFormat='M/D/YY hh:mm:ss[.fff]';
```

Error variables

The values of all error variables will exist after the script execution. The first variable, `ErrorMode`, is input from the user, and the last three are output from Qlik Sense with information on errors in the script.

Error variables overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

ErrorMode

This error variable determines what action is to be taken by Qlik Sense when an error is encountered during script execution.

ErrorMode

ScriptError

This error variable returns the error code of the last executed script statement.

ScriptError

ScriptErrorCount

This error variable returns the total number of statements that have caused errors during the current script execution. This variable is always reset to 0 at the start of script execution.

ScriptErrorCount

ScriptErrorList

This error variable will contain a concatenated list of all script errors that have occurred during the last script execution. Each error is separated by a line feed.

ScriptErrorList

ErrorMode

This error variable determines what action is to be taken by Qlik Sense when an error is encountered during script execution.

Syntax:

ErrorMode

Arguments:

Argument	Description
ErrorMode=1	The default setting. The script execution will halt and the user will be prompted for action (non-batch mode).
ErrorMode=0	Qlik Sense will simply ignore the failure and continue script execution at the next script statement.
ErrorMode=2	Qlik Sense will trigger an "Execution of script failed..." error message immediately on failure, without prompting the user for action beforehand.

Example:

```
set ErrorMode=0;
```

ScriptError

This error variable returns the error code of the last executed script statement.

Syntax:

ScriptError

This variable will be reset to 0 after each successfully executed script statement. If an error occurs it will be set to an internal Qlik Sense error code. Error codes are dual values with a numeric and a text component. The following error codes exist:

Error code	Description
0	No error
1	General error
2	Syntax error
3	General ODBC error
4	General OLE DB error
5	General custom database error
6	General XML error
7	General HTML error
8	File not found
9	Database not found
10	Table not found
11	Field not found
12	File has wrong format
13	BIFF error
14	BIFF error encrypted
15	BIFF error unsupported version
16	Semantic error

Example:

```
set ErrorMode=0;
LOAD * from abc.qvf;
if ScriptError=8 then
exit script;
//no file;
end if
```

ScriptErrorCount

This error variable returns the total number of statements that have caused errors during the current script execution. This variable is always reset to 0 at the start of script execution.

Syntax:

ScriptErrorCount**ScriptErrorList**

This error variable will contain a concatenated list of all script errors that have occurred during the last script execution. Each error is separated by a line feed.

Syntax:**ScriptErrorList**

2.5 Script expressions

Expressions can be used in both the **LOAD** statement and the **SELECT** statement. The syntax and functions described here apply to the **LOAD** statement, and not to the **SELECT** statement, since the latter is interpreted by the ODBC driver and not by Qlik Sense. However, most ODBC drivers are often capable of interpreting a number of the functions described below.

Expressions consist of functions, fields and operators, combined in a syntax.

All expressions in a Qlik Sense script return a number and/or a string, whichever is appropriate. Logical functions and operators return 0 for False and -1 for True. Number to string conversions and vice versa are implicit. Logical operators and functions interpret 0 as False and all else as True.

The general syntax for an expression is:

```

expression ::= (constant  constant          |
                fieldref    fieldref         |
                operator1 expression  operator1 expression |
                expression operator2 expression operator2 expression |
                function      function       |
                ( expression )          ( expression ) )

```

where:

constant is a string (a text, a date or a time) enclosed by single straight quotation marks, or a number. Constants are written with no thousands separator and with a decimal point as decimal separator.

fieldref is a field name of the loaded table.

operator1 is a unary operator (working on one expression, the one to the right).

operator2 is a binary operator (working on two expressions, one on each side).

function ::= functionname(parameters)

parameters ::= expression { , expression }

The number and types of parameters is not arbitrary. It depends on the function used.

Expressions and functions can thus be nested freely, and as long as the expression returns an interpretable value, Qlik Sense will not give any error messages.

3 Visualization expressions

An expression is a combination of functions, fields, and mathematical operators (+ * / =). Expressions are used to process data in the app in order to produce a result that can be seen in a visualization. They are not limited to use in measures. You can build visualizations that are more dynamic and powerful, with expressions for titles, subtitles, footnotes, and even dimensions.

This means, for example, that instead of the title of a visualization being static text, it can be made from an expression whose result changes depending on the selections made.



For detailed reference regarding script functions and chart functions, see the Qlik Sense online help.

See also:

- ▢ [Script expressions \(page 119\)](#)


3.1 Defining the aggregation scope

Normally, there are two different restrictions that together determine which records are relevant to an aggregation, namely the:

- Dimensional value (if in aggregating in a chart)
- Selection

Together, these restrictions define the aggregation scope. You may come across situations where you want to your calculation to disregard the selection, the dimension or both. In chart functions, you can achieve this by using the TOTAL qualifier, set analysis or a combination of the TOTAL qualifier and set analysis.

Method	Description
TOTAL qualifier	<p>Using the total qualifier inside your aggregation function disregards the dimensional value.</p> <p>The aggregation will be performed on all possible field values.</p> <p>The TOTAL qualifier may be followed by a list of one or more field names within angle brackets. These field names should be a subset of the chart dimension variables. In this case, the calculation is made disregarding all chart dimension variables except those listed, that is, one value is returned for each combination of field values in the listed dimension fields. Also, fields that are not currently a dimension in a chart may be included in the list. This may be useful in the case of group dimensions, where the dimension fields are not fixed. Listing all of the variables in the group causes the function to work when the drill-down level changes.</p>
Set	Using set analysis inside your aggregation disregards the selection. The aggregation will be

Method	Description
analysis	performed on all values split across the dimensions.
TOTAL qualifier and set analysis	Using set analysis inside your aggregation disregards the selection and the dimensions. <div style="border: 1px solid gray; padding: 5px; margin-top: 10px;">  <i>This method corresponds to using the ALL qualifier.</i> </div>

Example: TOTAL qualifier

The following example shows how TOTAL can be used to calculate a relative share. Assuming that Q2 has been selected, using TOTAL calculates the sum of all values disregarding the dimensions.

Year	Quarter	Sum(Amount)	Sum(TOTAL Amount)	Sum(Amount)/Sum(TOTAL Amount)
		3000	3000	100%
2012	Q2	1700	3000	56,7%
2013	Q2	1300	3000	43,3%



*To show the numbers as a percentage, in the properties panel, for the measure you want to show as a percentage value, under **Number formatting**, select **Number**, and from **Formatting**, choose **Simple** and one of the % formats.*

Example: Set analysis

The following example shows how set analysis can be used to make a comparison between data sets before any selection was made. Assuming that Q2 has been selected, using set analysis with the set definition {1} calculates the sum of all values disregarding any selections but split by the dimensions.

Year	Quarter	Sum(Amount)	Sum({1} Amount)	Sum(Amount)/Sum({1} Amount)
		3000	10800	27,8%
2012	Q1	0	1100	0%
2012	Q3	0	1400	0%
2012	Q4	0	1800	0%
2012	Q2	1700	1700	100%
2013	Q1	0	1000	0%
2013	Q3	0	1100	0%
2013	Q4	0	1400	0%
2013	Q2	1300	1300	100%

Example: TOTAL qualifier and set analysis

The following example shows how set analysis and the TOTAL qualifier can be combined to make a comparison between data sets before any selection was made and across all dimensions. Assuming that Q2 has been selected, using set analysis with the set definition {1} and the TOTAL qualifier calculates the sum of all values disregarding any selections and disregarding the dimensions.

Year	Quarter	Sum (Amount)	Sum({1} TOTAL Amount)	Sum(Amount)/Sum({1} TOTAL Amount)
		3000	10800	27,8%
2012	Q2	1700	10800	15,7%
2013	Q2	1300	10800	12%

Data used in examples:

```
AggregationScope:  
LOAD * inline [  
Year Quarter Amount  
2012 Q1 1100  
2012 Q2 1700  
2012 Q3 1400  
2012 Q4 1800  
2013 Q1 1000  
2013 Q2 1300  
2013 Q3 1100  
2013 Q4 1400] (delimiter is ' ');
```

3.2 Analyzing sets of data - Set analysis

Aggregation functions by default aggregate over the current selection of field values. The current selection can be referred to as a set of field values. You can define other sets of field values and use them in your visualizations instead of the current selection. For example, in a dashboard, you may want to show market share of a product across all regions, irrespective of the current selections.

Defining a set of field values is referred to as defining a set expression, and using set expressions to analyze data is referred to as set analysis.



Set expressions are available for visualizations only, not in scripts.

Set expressions always begin and end in curly brackets. For example, `sum({1} sales)`, where `{1}` is a set expression.

Building a set expression

In a set expression you must always identify how the set of field values you are defining relates to the field or expression you are evaluating. For example, are you evaluating the complete set of field values or the inverse of the current selection? After you have identified this relationship you can modify the selection of values within the field (this is optional).

In short, a set expression includes an identifier and an optional modifier. Modifiers are separated from identifiers by angled brackets as follows:

```
{set_identifier<set_modifier>}
```

You can use operators on both identifiers and modifiers to manipulate field relationships and selections. Additionally, Qlik Sense enables you to combine modifiers with dollar-sign expansions, advanced searches and implicit field value definitions as described in the following topics.

Example:

To understand the basics of set expression, let's look at a simple use case. We want to build a dashboard showing the following visualizations of value sales in the USA:

1. Total value sales in the USA by product group irrespective of current selection
2. Value sales in the USA by product group given the current selection

Our sales data is global but is divided in Region. Product groups are found in the ProductGroup field while value sales values are found in the Sales field.

In normal circumstances you could build one visualization with the dimension ProductGroup and the measure sum(Sales). User selections of Region and ProductGroup would then determine what is shown. However here we want visualization (1) above to always show the same region and selections and visualization (2) to always show the same region. Let's express what we want to see in each visualization in terms of a set expression:

1. set_expression= {all of Sales <for RegionUSA>}
2. set_expression= {current selection of Sales< for RegionUSA>}

Given that the:

- qualifier for "all" is **1**
- qualifier for **current selection** is **\$**
- syntax for modifiers in this case is `set_modifier = <field_name={field_value, [field_value]}`


we get the following measures for our visualizations using set expressions:

1. sum({1<Region={USA}>} Sales)
2. sum({\$<Region={USA}>} Sales)

Set identifiers

Set identifiers define the relationship between the set expression and the field values or expression that is being evaluated.

Set identifiers can be combined using set operators.

Identifier	Description
1	Represents the full set of all the records in the application.
\$	<p>Represents the records of the current selection.</p> <p>The set expression {} is thus the equivalent of not stating a set expression.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">  <i>{1-\$} is all the more interesting as it defines the inverse of the current selection, that is, everything that the current selection excludes.</i> </div>
\$N	<p>Selections from the Back stack can be used as set identifiers, by use of the dollar symbol: \$1 represents the previous selection, that is, it is equivalent to pressing the Back button.</p> <p>Any unsigned integer can be used in the Back notation. \$0 represents the current selection.</p>
\$_N	<p>Selections from the Forward stack can be used as set identifiers, by use of the dollar symbol: \$_1 represents one step forward, that is the equivalent to pressing the Forward button.</p> <p>Any unsigned integer can be used in the Forward notation. \$0 represents the current selection.</p>
bookmark_id bookmark_name	<p>Server and app bookmarks can be used as set identifiers. Either the bookmark ID or the bookmark name can be used. For example BM01 or MyBookMark.</p> <p>Only the selection part of a bookmark is used. The values are not included. It is thus not possible to use input fields in bookmarks for set analysis.</p>

Examples and results:

Examples	Results
sum({\$} Sales)	Returns sales for the current selection, that is to say, the same as sum(Sales)
sum({\$1} Sales)	Returns sales for the previous selection
sum({\$_2} Sales)	Returns sales for the 2nd next selection, that is to say, two steps forward. Only relevant if you just made two Back operations.

Examples	Results
sum({1} Sales)	Returns total sales within the application, disregarding the selection but not the dimension. If used in a chart with, for example, Products as dimension, each product will get a different value.
sum({1} Total Sales)	Returns total sales within the application, disregarding both selection and dimension, that is to say, the same as sum(All Sales) .
sum({BM01} Sales)	Returns sales for the bookmark BM01
sum({MyBookMark} Sales)	Returns sales for the bookmark MyBookMark
sum({Server\BM01} Sales)	Returns the sales for the server bookmark BM01
sum({App\MyBookmark} Sales)	Returns the sales for the app bookmark MyBookMark

See also:

- ▢ [Set operators \(page 126\)](#)

Set operators

There are several set operators that can be used in set expressions. All set operators use sets as operands and return a set as result.

The order of precedence is:

1. Unary minus (complement)
2. Intersection and Symmetric difference
3. Union and Exclusion.

Within a group, the expression is evaluated from left to right. Alternative orders can be defined by standard brackets, which may be necessary because the set operators do not commute. For example, **A+(B-C)** is different from **(A+B)-C** which in turn is different from **(A-C)+B**.



*The use of set operators in combination with basic aggregation expressions involving fields from multiple Qlik Sense tables may cause unpredictable results and should be avoided. E.g. if Quantity and Price are fields from different tables, then the expression `sum({$*BM01} Quantity*Price)` should be avoided.*

Arguments:

Operator	Description
+	Union. This binary operation returns a set consisting of the records that belong to any of the two set operands.
-	Exclusion. This binary operation returns a set of the records that belong to the first but not the other of the two set operands. Also, when used as a unary operator, it returns the complement set.
*	Intersection. This binary operation returns a set consisting of the records that belong to both of the two set operands.
/	Symmetric difference (XOR). This binary operation returns a set consisting of the records that belong to either, but not both of the two set operands.

Examples and results:

Examples	Results
<code>sum({1-\$} Sales)</code>	Returns sales for everything excluded by the current selection
<code>sum({\$*BM01} Sales)</code>	Returns sales for the intersection between the current selection and bookmark BM01
<code>sum({-(\$+BM01)} Sales)</code>	Returns sales excluded by current selection and bookmark BM01

Set modifiers

A set can be modified by an additional or a changed selection. Such a modification can be written in the set expression.

The modifier consists of one or several field names, each followed by a selection that should be made on the field, all enclosed by < and >. For example: <Year={2007,+2008},Region={US}>. Field names and field values can be quoted as usual, e.g. <[Sales Region]='west coast', 'South America'>.

A set modifier can be used on a set identifier or on its own. It cannot be used on a set expression. When used on a set identifier, the modifier must be written immediately after the set identifier, for examples `{$<Year = {2007, 2008}>}`. When used on its own, it is interpreted as a modification of the current selection.

There are several ways to define the selection as described in the following.

Based on another field

A simple case is a selection based on the selected values of another field, for example <OrderDate = DeliveryDate>. This modifier will take the selected values from **DeliveryDate** and apply those as a selection on **OrderDate**. If there are many distinct values – more than a couple of hundred – then this operation is CPU intense and should be avoided.

Based on element sets (a field value list in the modifier)


The most common case is a selection based on a field value list enclosed in curly brackets, the values separated by commas, for example `<Year = {2007, 2008}>`. The curly brackets here define an element set, where the elements can be either field values or searches of field values. A search is always defined by the use of double quotes, for example `<Ingredient = {"*Garlic*"}` will select all ingredients including the string 'garlic'. Searches are case-insensitive and are made also over excluded values.

Empty element sets, either explicitly for example `<Product = {}>` or implicitly for example `<Product = {"Perpetuum Mobile"}>` (a search with no hits) mean no product, i.e. they will result in a set of records that are not associated with any product. Note that this set cannot be achieved through usual selections, unless a selection is made in another field, for example. **TransactionID**.

Forced exclusion

Finally, for fields in and-mode, there is also the possibility of forced exclusion. If you want to force exclusion of specific field values, you will need to use "~" in front of the field name.

Examples and results:

Examples	Results
<code>sum({1<Region= {USA} >} Sales)</code>	Returns the sales for the region USA disregarding the current selection
<code>sum({\$<Region = >} Sales)</code>	Returns the sales for the current selection, but with the selection in 'Region' removed
<code>sum({<Region = >} Sales)</code>	Returns the same as the example immediately above. When the set to modify is omitted, \$ is assumed.
<div style="border: 1px solid gray; padding: 10px; width: fit-content; margin: 0 auto;">  <p><i>The syntax in the two previous examples is interpreted as “no selections” in 'Region', that is to say all regions given other selections will be possible. It is not equivalent to the syntax <code><Region = {}></code> (or any other text on the right side of the equal sign implicitly resulting in an empty element set) which is interpreted as no region.</i></p> </div>	
<code>sum({\$<Year = {2000}, Region = {US, SE, DE, UK, FR}>} Sales)</code>	Returns the sales for current selection, but with new selections both in 'Year' and in 'Region' .
<code>sum({\$<~Ingredient = {"*garlic*">}</code>	Returns the sales for current selection, but with a forced exclusion of all ingredients containing the string 'garlic' .

Examples	Results
Sales)	
sum({\$<Year = {"2*"}>} Sales)	Returns the sales for the current selection, but with all years beginning with the digit "2", i.e. most likely year 2000 and onwards, selected in the field 'Year'.
sum({\$<Year = {"2*", "198*"}>} Sales)	As above, but now also the 1980:s are included in the selection.
sum({\$<Year = {">1978<2004"}>} Sales)	As above, but now with a numeric search so that an arbitrary range can be specified.

Set modifiers with set operators

The selection within a field can be defined using set operators working on different element sets. For example the modifier `<Year = {"20*", 1997} - {2000}>` will select all years beginning with "20" in addition to "1997", except for "2000".

Examples and results:

Examples	Results
sum({\$<Product = Product + {OurProduct1} - {OurProduct2}>} Sales)	Returns the sales for the current selection, but with the product "OurProduct1" added to the list of selected products and "OurProduct2" removed from the list of selected products.
sum({\$<Year = Year + {"20*", 1997} - {2000}>} Sales)	Returns the sales for the current selection but with additional selections in the field "Year": 1997 and all that begin with "20" – however, not 2000. Note that if 2000 is included in the current selection, it will still be included after the modification.
sum({\$<Year = (Year + {"20*", 1997}) - {2000}>} Sales)	Returns almost the same as above, but here 2000 will be excluded, also if it initially is included in the current selection. The example shows the importance of sometimes using brackets to define an order of precedence.
sum({\$<Year = {"*"} - {2000}, Product = {"*bearing*"}>} Sales)	Returns the sales for the current selection but with a new selection in "Year": all years except 2000; and only for products containing the string 'bearing'.

Set modifiers using assignments with implicit set operators

This notation defines new selections, disregarding the current selection in the field. However, if you want to base your selection on the current selection in the field and add field values, for example you may want a modifier `<Year = Year + {2007, 2008}>`. A short and equivalent way to write this is `<Year += {2007, 2008}>`, i.e. the assignment operator implicitly defines a union. Also implicit intersections, exclusions and symmetric differences can be defined using `"*="`, `"-="` and `"/=`.

Examples and results:

Examples	Results
sum({\$<Product += {OurProduct1, OurProduct2} >} Sales)	Returns the sales for the current selection, but using an implicit union to add the products 'OurProduct1' and 'OurProduct2' to the list of selected products.
sum({\$<Year += {"20*", 1997} – {2000} >} Sales)	Returns the sales for the current selection but using an implicit union to add a number of years in the selection: 1997 and all that begin with "20" – however, not 2000. Note that if 2000 is included in the current selection, it will still be included after the modification. Same as <Year=Year + ({"20*", 1997}–{2000})>.
sum({\$<Product *= {OurProduct1} >} Sales)	Returns the sales for the current selection, but only for the intersection of currently selected products and the product OurProduct1.

Set modifiers with advanced searches

Advanced searches using wild cards and aggregations can be used to define sets.

Examples and results:

Examples	Results
sum({\$–1<Product = {"*Internal*", "*Domestic*"}>} Sales)	Returns the sales for current selection, excluding transactions pertaining to products with the string 'Internal' or 'Domestic' in the product name.
sum({\$<Customer = {"=Sum ({1<Year = {2007}>} Sales) > 1000000"}>} Sales)	Returns the sales for current selection, but with a new selection in the 'Customer' field: only customers who during 2007 had a total sales of more than 1000000.

Set modifiers with dollar-sign expansions

Variables and other dollar-sign expansions can be used in set expressions.

Examples and results:

Examples	Results
sum({\$<Year = {\$(#vLastYear)}>} Sales)	Returns the sales for the previous year in relation to current selection. Here, a variable vLastYear containing the relevant year is used in a dollar-sign expansion.
sum({\$<Year = {\$(#=Only(Year)-1)}>} Sales)	Returns the sales for the previous year in relation to current selection. Here, a dollar-sign expansion is used to calculate previous year.

Set modifiers with implicit field value definitions

The following describes how to define a set of field values using a nested set definition.

In such cases, the element functions P() and E() must be used, representing the element set of possible values and the excluded values of a field, respectively. Inside the brackets, it is possible to specify one set expression and one field, for example P({1} customer). These functions cannot be used in other expressions:

Examples and results:

Examples	Results
<pre>sum({\$<Customer = P({1<Product= {'Shoe'}>} Customer)>} Sales)</pre>	Returns the sales for current selection, but only those customers that ever have bought the product 'Shoe'. The element function P() here returns a list of possible customers; those that are implied by the selection 'Shoe' in the field Product.
<pre>sum({\$<Customer = P({1<Product= {'Shoe'}>})>} Sales)</pre>	Same as above. If the field in the element function is omitted, the function will return the possible values of the field specified in the outer assignment.
<pre>sum({\$<Customer = P({1<Product= {'Shoe'}>} Supplier)>} Sales)</pre>	Returns the sales for current selection, but only those customers that ever have supplied the product 'Shoe'. The element function P() here returns a list of possible suppliers; those that are implied by the selection 'Shoe' in the field Product. The list of suppliers is then used as a selection in the field Customer.
<pre>sum({\$<Customer = E({1<Product= {'Shoe'}>})>} Sales)</pre>	Returns the sales for current selection, but only those customers that never bought the product 'Shoe'. The element function E() here returns the list of excluded customers; those that are excluded by the selection 'Shoe' in the field Product.

Syntax for sets

The full syntax (not including the optional use of standard brackets to define precedence) is described using Backus-Naur Formalism:

```
set_expression ::= { set_entity { set_operator set_entity } }
set_entity ::= set_identifier [ set_modifier ]
set_identifier ::= 1 | $ | $N | $_N | bookmark_id | bookmark_name
set_operator ::= + | - | * | /
set_modifier ::= < field_selection { , field_selection } >
```

```
field_selection ::= field_name [ = | += | -= | *= | /= ] element_set_
expression
element_set_expression ::= element_set { set_operator element_set }
element_set ::= [ field_name ] | { element_list } | element_function
element_list ::= element { , element }
element_function ::= ( P | E ) ( [ set_expression ] [ field_name ] )
element ::= field_value | " search_mask "
```

See also:

- ▢ [What is Backus-Naur formalism? \(page 13\)](#)

3.3 Syntax

The syntax used for chart expressions and aggregations is described in the following sections.

General syntax for chart expressions

expression ::= (constant	
expressionname	
operator1 expression	
expression operator2 expression	
function	
aggregation function	
(expression))

where:

constant is a string (a text, a date or a time) enclosed by single straight quotation marks, or a number. Constants are written without thousands separator and with a decimal point as decimal separator.

expressionname is the name (label) of another expression in the same chart.

operator1 is a unary operator (working on one expression, the one to the right).

operator2 is a binary operator (working on two expressions, one on each side).

function ::= functionname (parameters)

parameters ::= expression { , expression }

The number and types of parameters are not arbitrary. They depend on the function used.

aggregationfunction ::= aggregationfunctionname (parameters2)

parameters2 ::= aggexpression { , aggexpression }

The number and types of parameters are not arbitrary. They depend on the function used.

See also:

▢ [Operators \(page 134\)](#)

General syntax for aggregations

aggexpression ::= (fieldref	
operator1 aggexpression	
aggexpression operator2 aggexpression	
functioninaggr	
(aggexpression))

fieldref is a field name.

functionaggr ::= functionname (parameters2)

Expressions and functions can thus be nested freely, as long as **fieldref** is always enclosed by exactly one aggregation function and provided the expression returns an interpretable value, Qlik Sense does not give any error messages.

See also:

▢ [Operators \(page 134\)](#)

4 Operators

This section describes the operators that can be used in Qlik Sense. There are two types of operators:

- Unary operators (take only one operand)
- Binary operators (take two operands)

Most operators are binary.

The following operators can be defined:

- Bit operators
- Logical operators
- Numeric operators
- Relational operators
- String operators

4.1 Bit operators

All bit operators convert (truncate) the operands to signed integers (32 bit) and return the result in the same way. All operations are performed bit by bit. If an operand cannot be interpreted as a number, the operation will return NULL.

bitnot Bit inverse. Unary operator. The operation returns the logical inverse of the operand performed bit by bit.

Example:

`bitnot 17` returns -18

bitand Bit and. The operation returns the logical AND of the operands performed bit by bit.

Example:

`17 bitand 7` returns 8

bitor Bit or. The operation returns the logical OR of the operands performed bit by bit.

Example:

`17 bitor 7` returns 23

bitxor Bit exclusive or. The operation returns the logical exclusive or of the operands performed bit by bit.

Example:

`17 bitxor 7` returns 22

>> Bit right shift. The operation returns the first operand shifted to the right. The number of steps is defined in the second operand.

Example:

8 >> 2 returns 2

<< Bit left shift. The operation returns the first operand shifted to the left. The number of steps is defined in the second operand.

Example:

8 << 2 returns 32

4.2 Logical operators

All logical operators interpret the operands logically and return True (-1) or False (0) as result.

not Logical inverse. One of the few unary operators. The operation returns the logical inverse of the operand.

and Logical and. The operation returns the logical and of the operands.

or Logical or. The operation returns the logical or of the operands.

Xor Logical exclusive or. The operation returns the logical exclusive or of the operands. I.e. like logical or, but with the difference that the result is False if both operands are True.

4.3 Numeric operators

All numeric operators use the numeric values of the operands and return a numeric value as result.

+ Sign for positive number (unary operator) or arithmetic addition. The binary operation returns the sum of the two operands.

- Sign for negative number (unary operator) or arithmetic subtraction. The unary operation returns the operand multiplied by -1, and the binary the difference between the two operands.

***** Arithmetic multiplication. The operation returns the product of the two operands.

/ Arithmetic division. The operation returns the ratio between the two operands.

4.4 Relational operators

All relational operators compare the values of the operands and return True (-1) or False (0) as the result. All relational operators are binary.

<	Less than	A numeric comparison is made if both operands can be interpreted numerically. The operation returns the logical value of the evaluation of the comparison.
<=	Less than or equal	A numeric comparison is made if both operands can be interpreted numerically. The operation returns the logical value of the evaluation of the comparison.
>	Greater than	A numeric comparison is made if both operands can be interpreted numerically. The operation returns the logical value of the evaluation of the comparison.
>=	Greater than or equal	A numeric comparison is made if both operands can be interpreted numerically. The operation returns the logical value of the evaluation of the comparison.
=	Equals	A numeric comparison is made if both operands can be interpreted numerically. The operation returns the logical value of the evaluation of the comparison.
<>	Not equivalent to	A numeric comparison is made if both operands can be interpreted numerically. The operation returns the logical value of the evaluation of the comparison.
precedes	ASCII less than	Unlike the < operator no attempt is made to make a numeric interpretation of the argument values before the comparison. The operation returns true if the value to the left of the operator has a text representation which, in ASCII comparison, comes before the text representation of the value on the right.

Example:

' 11' precedes ' 2' returns True

compare this to:

' 11' < ' 2' returns False

follows	ASCII greater than	Unlike the > operator no attempt is made to make a numeric interpretation of the argument values before the comparison. The operation returns true if the value to the left of the operator has a text representation which, in ASCII comparison, comes after the text representation of the value on the right.
----------------	--------------------	--

Example:

' 23' follows ' 111' returns True

compare this to:

' 23' > ' 111' returns False

4.5 String operators

There are two string operators. One uses the string values of the operands and return a string as result. The other one compares the operands and returns a boolean value to indicate match.

& String concatenation. The operation returns a text string, that consists of the two operand strings, one after another.

Example:

'abc' & 'xyz' returns 'abcxyz'

like String comparison with wildcard characters. The operation returns a boolean True (-1) if the string before the operator is matched by the string after the operator. The second string may contain the wildcard characters * (any number of arbitrary characters) or ? (one arbitrary character).

Example:

'abc' like 'a*' returns True (-1)

'abcd' like 'a?c*' returns True (-1)

'abc' like 'a??bc' returns False (0)

5 Functions in scripts and chart expressions

This section describes functions that can be used in Qlik Sense data load scripts and chart expressions to transform and aggregate data.

Many functions can be used in the same way in both data load scripts and chart expressions, but there are a number of exceptions:

- Some functions can only be used in data load scripts, denoted by - script function.
- Some functions can only be used in chart expressions, denoted by - chart function.
- Some functions can be used in both data load scripts and chart expressions, but with differences in parameters and application. These are described in separate topics denoted by - script function or - chart function.

5.1 Aggregation functions

An aggregation function aggregates over the set of possible records defined by the selection, and returns a single value describing a property of several records in the data, for example a sum or a count.

Most aggregation functions can be used in both the data load script and chart expressions, but the syntax differs.

Using aggregation functions in a data load script

Aggregation functions can only be used in field lists for **LOAD** statements with a **group by** clause.

Using aggregation functions in chart expressions

The argument expression of one aggregation function must not contain another aggregation function.

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

An aggregation function aggregates over the set of possible records defined by the selection. However, an alternative set of records can be defined by using a set expression in set analysis.

Aggr - chart function

Aggr() returns an array of values of the expression calculated over dimensions. The **Aggr** function is used for advanced aggregations.

Basic aggregation functions, such as **Sum**, **Min**, and **Avg**, return a single numerical value while the result of an advanced aggregation can be compared to a temporary straight table that can be used in charts. To obtain a final aggregation from this temporary table, the **Aggr** function should be placed inside a basic aggregation function, for example, **Sum**, **Max** or **Count**.

5 Functions in scripts and chart expressions



Use this function in calculated dimensions if you want to create nested chart aggregation in multiple levels.

Syntax:

```
Aggr ({[DISTINCT] [NODISTINCT ]} expr, dim{, Expression})
```

Return data type: dual

Arguments:

Argument	Description
expr	By default, the aggregation function will aggregate over the set of possible records defined by the selection.
dim	Single field. Cannot be an expression.
Expression	Optional expressions or fields containing the range of data to be measured.
DISTINCT	If the expression argument is preceded by the distinct qualifier or if no qualifier is used at all, each distinct combination of dimension values will generate only one return value. This is the normal way aggregations are made – each distinct combination of dimension values will render one line in the chart.
NODISTINCT	If the expression argument is preceded by the nodistinct qualifier, each combination of dimension values may generate more than one return value, depending on underlying data structure. If there is only one dimension, the aggr function will return an array with the same number of elements as there are rows in the source data.

Limitations:

Each dimension must be a single field, and cannot be an expression (calculated dimension).

Examples and results:

Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20

5 Functions in scripts and chart expressions

Customer	Product	UnitSales	UnitPrice
Betacab	DD	25	25
Canutility	AA	8	15
Canutility	CC	-	19

Create a table with **Customer**, **Product**, **UnitPrice**, and **UnitSales** as dimensions.

Example	Result
Aggr(Max (UnitPrice), Customer)	<p>An array of values: 16, 20, 15, and 25. The expression finds the maximum UnitPrice by Customer.</p> <p>The aggregation Max(UnitPrice) produces a result for each Product by Customer. By using this expression as the expr argument in the Aggr() function and Customer as the dim argument, we can find the result of Max(UnitPrice) by Customer.</p>
Min(Aggr(Max (UnitPrice), Customer))	<p>15. The expression finds the maximum UnitPrice by Customer, and finds the minimum value of the result.</p> <p>By using the Aggr expression as input to the Min() function, the minimum value of the array produced by the Aggr() is found. Effectively, by enclosing the Aggr() function in another aggregation we have built a temporary list of values without having to create a separate chart containing those values.</p>
Aggr (NODISTINCT Max (UnitPrice), Customer)	<p>An array of values: 16, 16, 16, 25, 25, 25, 15, 15, 25, and 25. The nodistinct qualifier means that the array contains one element for each row in the source data: each is the maximum UnitPrice for each Customer and Product.</p>

Data used in examples:

```
Temp:
LOAD * inline [
Customer Product UnitSales UnitPrice
Astrida AA 4 16
Astrida AA 10 15
Astrida BB 9 9
Betacab BB 5 10
Betacab CC 2 20
Betacab DD 1 25 25
Canutility AA 8 15
Canutility CC 19
] (delimiter is ' ');
```

Basic aggregation functions

Basic aggregation functions overview

Basic aggregation functions are a group of the most common aggregation functions.

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Basic aggregation functions in the data load script

firstsortedvalue

This script function returns the first value of an expression sorted by corresponding **sort-weight** when the expression is iterated over a number of records as defined by a **group by** clause.

```
firstsortedvalue ([ distinct ] expression [, sort-weight [, n ]])
```

max

This script function returns the maximum numeric value of expression encountered over a number of records as defined by a **group by** clause.

```
max ( expression[, rank])
```

min

This script function returns the minimum numeric value of expression encountered over a number of records as defined by a **group by** clause.

```
min ( expression[, rank])
```

mode

This script function returns the mode value, i.e. the most commonly occurring value, of expression over a number of records, as defined by a **group by** clause. **mode** can return numeric values as well as text values.

```
mode (expression )
```

only

This script function returns the value of an expression or field iterated over the one or more records. If records contain only one value then that value is returned, otherwise NULL is returned. Use **group by** clause to evaluate over multiple records. **only** can return numeric and text values.

```
only (expression )
```

sum

This script function returns the sum of expression over a number of records as defined by a **group by** clause.

```
sum ([distinct]expression)
```

Basic aggregation functions in chart expressions

Chart aggregation functions can only be used on fields in chart expressions. The argument expression of one aggregation function must not contain another aggregation function.

FirstSortedValue

FirstSortedValue() returns the value of one field based on the sorted values of another field. For example, the product with the lowest unit price.

```
FirstSortedValue - chart function([SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]] value, sort_weight [,rank])
```

Max

Max() finds the highest value of the aggregated data. By specifying a **rank** n, the nth highest value can be found.

```
Max - chart functionMax() finds the highest value of the aggregated data. By specifying a rank n, the nth highest value can be found. You might also want to look at FirstSortedValue and rangemax, which have similar functionality to the Max function. Max([SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]] expr [,rank])Return data
```

type:numeric ArgumentDescriptionexprThe expression or field containing the data to be measured.rankThe default value of rank is 1, which corresponds to the highest value. By specifying rank as 2, the second highest value is returned. If rank is 3, the third highest value is returned, and so on.SetExpressionBy default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression. TOTALIf the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables. CustomerProductUnitSalesUnitPrice AstridaAA416AstridaAA1015AstridaBB99BetacabBB510BetacabCC220BetacabDD-25CanutilityAA815CanutilityCC-19ExamplesResultsMax(UnitSales)10, because this is the highest value in UnitSales.The value of an order is calculated from the number of units sold in (UnitSales) multiplied by the unit price.Max(UnitSales*UnitPrice)150, because this is the highest value of the result of calculating all possible values of (UnitSales)*(UnitPrice).Max (UnitSales, 2)9, which is the second highest value.Max(TOTAL UnitSales)10, because the TOTAL qualifier means the highest possible value is found, disregarding the chart dimensions. For a chart with Customer as dimension, the TOTAL qualifier will ensure the maximum value across the full dataset is returned, instead of the maximum UnitSales for each customer.Make the selection Customer B.Max({1} TOTAL UnitSales)10, independent of the

5 Functions in scripts and chart expressions

```
selection made, because the Set Analysis expression {1} defines the set of records to be evaluated as ALL, no matter what selection is made. Data used in examples: ProductData:LOAD * inline
[Customer|Product|UnitSales|UnitPriceAstrida|AA|4|16Astrida|AA|10|15Astrida|BB|9|9Betacab|BB|5|10Betacab|CC|2|20Betacab|DD||25Canutility|AA|8|15Canutility|CC||19] (delimiter is '|'); FirstSortedValue RangeMax
({SetExpression}) [DISTINCT] [TOTAL [<fld {,fld}>]] expr [,rank])
```

Min

Min() finds the lowest value of the aggregated data. By specifying a **rank** n, the nth lowest value can be found.

```
Min - chart function({SetExpression}) [DISTINCT] [TOTAL [<fld {,fld}>]]
expr [,rank])
```

Mode

Mode() finds the most commonly-occurring value, the mode value, in the aggregated data. The **Mode()** function can process text values as well as numeric values.

```
Mode - chart function({SetExpression} [TOTAL [<fld {,fld}>]]) expr)
```

Only

Only() returns a value if there is one and only one possible result from the aggregated data. For example, searching for the only product where the unit price =9 will return NULL if more than one product has a unit price of 9.

```
Only - chart function({SetExpression}) [DISTINCT] [TOTAL [<fld {,fld}>]]
expr)
```

Sum

Sum() calculates the total of the values given by the expression or field across the aggregated data.

```
Sum - chart function({SetExpression}) [DISTINCT] [TOTAL [<fld {,fld}>]]
expr)
```

firstsortedvalue

This script function returns the first value of an expression sorted by corresponding **sort-weight** when the expression is iterated over a number of records as defined by a **group by** clause.

Syntax:

```
firstsortedvalue ([ distinct ] value, sort-weight [, rank ])
```

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
value	The function returns the value from the field specified in value associated with the result of sorting the sort_weight field, taking into account rank , if specified. If more than one resulting value shares the same sort_weight for the specified rank , the function returns NULL.
sort-weight	The expression containing the data to be sorted. The first (lowest) value of sort_weight is found, from which the corresponding value of the value expression is determined. If you place a minus sign in front of sort_weight , the function returns the last (highest) sorted value instead. .
rank Expression	By stating a rank "n" larger than 1, you get the nth sorted value.
distinct	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Example	Result
<p>Temp:</p> <pre>LOAD * inline [Customer Product OrderNumber UnitSales CustomerID Astrida AA 1 10 1 Astrida AA 7 18 1 Astrida BB 4 9 1 Astrida CC 6 2 1 Betacab AA 5 4 2 Betacab BB 2 5 2 Betacab DD Canutility DD 3 8 Canutility CC] (delimiter is ' ');</pre> <p>FirstSortedValue:</p> <pre>LOAD Customer,FirstSortedValue(Product, - UnitSales) as MyProductWithLargestOrderByCustomer Resident Temp Group By Customer;</pre>	<pre>MyProductWithLargestOrderByCustomer AA BB DD</pre> <p>because AA corresponds to the largest order (value of UnitSales:18) for customer Astrida, BB corresponds to the largest order (5) for customer Betacab and DD corresponds to the largest order (8) for customer Canutility.</p>
<p>Given that the Temp table is loaded as in the previous example:</p> <pre>LOAD Customer,FirstSortedValue(Product, UnitSales) as MyProductWithSmallestOrderByCustomer Resident</pre>	<pre>MyProductWithSmallestOrderByCustomer CC AA DD</pre> <p>because CC corresponds to the smallest order (2) for customer Astrida, AA corresponds to the</p>

5 Functions in scripts and chart expressions

Example	Result
Temp Group By Customer;	smallest order (4) for customer Betacab and DD corresponds to the smallest order (8) for customer Canutility (there is only one valid order for customer Canutility so it is both the smallest and the largest).
<p>Given that the Temp table is loaded as in first example:</p> <pre>LOAD Customer, FirstSortedValue(Product, - UnitsSales,2) as My2ndProductOrderCustomer, Resident Temp Group By Customer;</pre>	<pre>MySecondLargetsOrderCustomer AA AA -</pre> <p>Note! The field will only show AA once, because it is the second-largest order for both customers Astrida and Betacab.</p>

FirstSortedValue - chart function

FirstSortedValue() returns the value of one field based on the sorted values of another field. For example, the product with the lowest unit price.

Syntax:

```
FirstSortedValue([SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]] value,
sort_weight [,rank])
```

Return data type: dual

Arguments:

Argument	Description
value	Output field. The function returns the value from the field specified in value associated with the result of sorting the sort_weight field, taking into account rank , if specified. If more than one resulting value shares the same sort_weight for the specified rank , the function returns NULL.
sort_weight	Input field. The expression containing the data to be sorted. The first (lowest) value of sort_weight is found, from which the corresponding value of the value expression is determined. If you place a minus sign in front of sort_weight , the function returns the last (highest) sorted value instead.
rank	By stating a rank "n" larger than 1, you get the nth sorted value.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.

5 Functions in scripts and chart expressions

Argument	Description
TOTAL	<p>If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions.</p> <p>The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.</p>

Examples and results:

Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

Example	Result
firstsortedvalue (Product, UnitPrice)	BB, which is the Product with the lowest unitPrice(9).
firstsortedvalue (Product, UnitPrice, 2)	BB, which is the Product with the second-lowest unitPrice(10).
firstsortedvalue (Customer, - UnitPrice, 2)	B, which is the Customer with the Product that has second-highest unitPrice(20).
firstsortedvalue (Customer, UnitPrice, 3)	NULL, because there are two values of customer (A and C) with the same rank (third-lowest) unitPrice(15).
firstsortedvalue (Customer, - UnitPrice*UnitSales, 2)	A, which is the customer with the second-highest sales order value unitPrice multiplied by unitSales (120).

Data used in examples:

```

ProductData:
LOAD * inline [
Customer|Product|UnitSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9

```

5 Functions in scripts and chart expressions

```
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD|1|25
Canutility|AA|8|15
Canutility|CC|1|19
] (delimiter is '|');
```

max

This script function returns the maximum numeric value of expression encountered over a number of records as defined by a **group by** clause.

Syntax:

```
max ( expression[, rank] )
```

Arguments:

Argument	Description
rank	The default value of rank is 1, which corresponds to the highest value. By specifying rank as 2, the second highest value is returned. If rank is 3, the third highest value is returned, and so on.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Example	Result
Temp: LOAD * inline [Customer Product OrderNumber UnitsSales CustomerID Astrida AA 1 10 1 Astrida AA 7 18 1 Astrida BB 4 9 1 Astrida CC 6 2 1 Betacab AA 5 4 2 Betacab BB 2 5 2 Betacab DD Canutility DD 3 8 Canutility CC] (delimiter is ' ');	Customer MyMax A 18 B 5 C 8
Max: LOAD Customer, Max(UnitsSales) as MyMax, Resident Temp Group By Customer;	
Given that the Temp table is loaded as in previous example:	Customer MyMaxRank2

5 Functions in scripts and chart expressions

Example	Result	
LOAD Customer, Max(UnitSales,2) as MyMaxRank2, Resident Temp Group By Customer;	A	10
	B	4
	C	-

Max - chart function

Max() finds the highest value of the aggregated data. By specifying a **rank** n, the nth highest value can be found.



You might also want to look at **FirstSortedValue** and **rangemax**, which have similar functionality to the **Max** function.

Syntax:

```
Max ( [{SetExpression}] [DISTINCT] [TOTAL [<fld {,fld}>]] expr [,rank] )
```

Return data type:numeric

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
rank	The default value of rank is 1, which corresponds to the highest value. By specifying rank as 2, the second highest value is returned. If rank is 3, the third highest value is returned, and so on.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Examples and results:

Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15

5 Functions in scripts and chart expressions

Customer	Product	UnitSales	UnitPrice
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

Examples	Results
Max(UnitSales)	10, because this is the highest value in unitSales.
The value of an order is calculated from the number of units sold in (unitSales) multiplied by the unit price. Max(UnitSales*UnitPrice)	150, because this is the highest value of the result of calculating all possible values of (unitSales)*(UnitPrice).
Max(UnitSales, 2)	9, which is the second highest value.
Max(TOTAL UnitSales)	10, because the TOTAL qualifier means the highest possible value is found, disregarding the chart dimensions. For a chart with Customer as dimension, the TOTAL qualifier will ensure the maximum value across the full dataset is returned, instead of the maximum UnitSales for each customer.
Make the selection Customer B. Max({1} TOTAL UnitSales)	10, independent of the selection made, because the Set Analysis expression {1} defines the set of records to be evaluated as ALL, no matter what selection is made.

Data used in examples:

```
ProductData:
LOAD * inline [
Customer|Product|UnitSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD||25
Canutility|AA|8|15
Canutility|CC||19
] (delimiter is '|');
```

5 Functions in scripts and chart expressions

See also:

- ▢ *FirstSortedValue - chart function (page 145)*
- ▢ *RangeMax (page 484)*

min

This script function returns the minimum numeric value of expression encountered over a number of records as defined by a **group by** clause.

Syntax:

```
min ( expression[, rank])
```

Arguments:

Argument	Description
rank	The default value of rank is 1, which corresponds to the lowest value. By specifying rank as 2, the second lowest value is returned. If rank is 3, the third lowest value is returned, and so on.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Example	Result								
<pre>Temp: LOAD * inline [Customer Product OrderNumber UnitSales CustomerID Astrida AA 1 10 1 Astrida AA 7 18 1 Astrida BB 4 9 1 Astrida CC 6 2 1 Betacab AA 5 4 2 Betacab BB 2 5 2 Betacab DD Canutility DD 3 8 Canutility CC] (delimiter is ' '); Min: LOAD Customer, Min(UnitSales) as MyMin, Resident Temp Group By Customer;</pre>	<table><tr><td>Customer</td><td>MyMin</td></tr><tr><td>A</td><td>2</td></tr><tr><td>B</td><td>4</td></tr><tr><td>C</td><td>8</td></tr></table>	Customer	MyMin	A	2	B	4	C	8
Customer	MyMin								
A	2								
B	4								
C	8								

5 Functions in scripts and chart expressions

Example	Result
Given that the Temp table is loaded as in previous example: LOAD Customer, Min(UnitSales,2) as MyMinRank2, Resident Temp Group By Customer;	Customer MyMinRank2 A 9 B 5 C -

Min - chart function

Min() finds the lowest value of the aggregated data. By specifying a **rank** n, the nth lowest value can be found.



You might also want to look at **FirstSortedValue** and **rangemin**, which have similar functionality to the **Min** function.

Syntax:

```
Min ([SetExpression] [TOTAL [<fld {, fld}>]]) expr [,rank])
```

Return data type: numeric

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
rank	The default value of rank is 1, which corresponds to the highest value. By specifying rank as 2, the second highest value is returned. If rank is 3, the third highest value is returned, and so on.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Examples and results:

Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16

5 Functions in scripts and chart expressions

Customer	Product	UnitSales	UnitPrice
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19



The `Min()` function must return a non-NULL value from the array of values given by the expression, if there is one. So in the examples, because there are NULL values in the data, the function returns the first non-NULL value evaluated from the expression.

Examples	Results
<code>Min(UnitSales)</code>	2, because this is the lowest non-NULL value in <code>unitSales</code> .
The value of an order is calculated from the number of units sold in (<code>unitSales</code>) multiplied by the unit price. <code>Min(UnitSales*UnitPrice)</code>	40, because this is the lowest non-NULL value result of calculating all possible values of (<code>unitSales</code>)*(<code>UnitPrice</code>).
<code>Min(UnitSales, 2)</code>	4, which is the second lowest value (after the NULL values).
<code>Min(TOTAL UnitSales)</code>	2, because the TOTAL qualifier means the lowest possible value is found, disregarding the chart dimensions. For a chart with Customer as dimension, the TOTAL qualifier will ensure the minimum value across the full dataset is returned, instead of the minimum UnitSales for each customer.
Make the selection Customer B. <code>Min({1} TOTAL UnitSales)</code>	40, independent of the selection made, because the Set Analysis expression <code>{1}</code> defines the set of records to be evaluated as ALL, no matter what selection is made.

Data used in examples:

```
ProductData:
LOAD * inline [
Customer|Product|UnitSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
```


5 Functions in scripts and chart expressions

```
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD||25
Canutility|AA|8|15
Canutility|CC||19
] (delimiter is '|');
```

See also:

- ▢ [FirstSortedValue - chart function \(page 145\)](#)
- ▢ [RangeMin \(page 487\)](#)

mode

This script function returns the mode value, i.e. the most commonly occurring value, of expression over a number of records, as defined by a **group by** clause. **mode** can return numeric values as well as text values.

Syntax:

```
mode ( expression )
```

Limitations:

If more than one value is equally commonly occurring, NULL is returned.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Example	Result
<pre>Temp: LOAD * inline [Customer Product OrderNumber UnitsSales CustomerID Astrida AA 1 10 1 Astrida AA 7 18 1 Astrida BB 4 9 1 Astrida CC 6 2 1 Betacab AA 5 4 2 Betacab BB 2 5 2 Betacab DD Canutility DD 3 8 Canutility CC] (delimiter is ' '); Mode: LOAD Customer, Mode(Product) as MyMostOftenSoldProduct, Resident Temp Group By Customer;</pre>	<pre>MyMostOftenSoldProduct AA because AA is the only product sold more than once.</pre>

Mode - chart function

Mode() finds the most commonly-occurring value, the mode value, in the aggregated data. The **Mode()** function can process text values as well as numeric values.

Syntax:

```
Mode ({[SetExpression] [TOTAL [<fld {, fld}>]]} expr)
```

Return data type: dual

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Examples and results:

Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

5 Functions in scripts and chart expressions

Examples	Results
Mode(UnitPrice) Make the selection Customer A.	15, because this is the most commonly-occurring value in unitsales. Returns NULL (-). No single value occurs more often than another.
Mode(Product) Make the selection Customer A	AA, because this is the most commonly occurring value in Product. Returns NULL (-). No single value occurs more often than another.
Mode (TOTAL UnitPrice)	15, because the TOTAL qualifier means the most commonly occurring value is still 15, even disregarding the chart dimensions.
Make the selection Customer B. Mode)({1} TOTAL UnitPrice)	15, independent of the selection made, because the Set Analysis expression {1} defines the set of records to be evaluated as ALL, no matter what selection is made.

Data used in examples:

```
ProductData:
LOAD * inline [
Customer|Product|UnitsSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD||25
Canutility|AA|8|15
Canutility|CC||19
] (delimiter is '|');
```

See also:

- [Avg - chart function \(page 190\)](#)
- [Median - chart function \(page 217\)](#)

only

This script function returns the value of an expression or field iterated over the one or more records. If records contain only one value then that value is returned, otherwise NULL is returned. Use **group by** clause to evaluate over multiple records. **only** can return numeric and text values.

Syntax:

```
only ( expression )
```

5 Functions in scripts and chart expressions

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Example	Result	
<pre>Temp: LOAD * inline [Customer Product OrderNumber UnitsSales CustomerID Astrida AA 1 10 1 Astrida AA 7 18 1 Astrida BB 4 9 1 Astrida CC 6 2 1 Betacab AA 5 4 2 Betacab BB 2 5 2 Betacab DD Canutility DD 3 8 Canutility CC] (delimiter is ' '); Only: LOAD Customer, Only(CustomerID) as MyUniqIDCheck, Resident Temp Group By Customer;</pre>	Customer A	MyUniqIDCheck 1 because only customer A has complete records that include CustomerID.

Only - chart function

Only() returns a value if there is one and only one possible result from the aggregated data. For example, searching for the only product where the unit price =9 will return NULL if more than one product has a unit price of 9.

Syntax:

```
Only ([{SetExpression}] [TOTAL [<fld {,fld}>]] expr)
```

Return data type: dual

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the

5 Functions in scripts and chart expressions

Argument	Description
	current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.



Use `Only()` when you want a `NULL` result if there are multiple possible values in the sample data.

Examples and results:

Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

Examples	Results
<code>Only({<UnitPrice={9}>} Product)</code>	BB, because this is the only Product that has a unitPrice of '9'.
<code>Only({<Product={DD}>} Customer)</code>	B, because the only customer selling a Product called 'DD'.
<code>Only({<UnitPrice={20}>} UnitSales)</code>	The number of unitSales where unitPrice is 20 is 2, because there is only one value of unitSales where the unitPrice =20.
<code>Only({<UnitPrice={15}>} unitSales)</code>	NULL, because there are two values of unitSales where the unitPrice =15.

Data used in examples:

```
ProductData:
LOAD * inline [
Customer|Product|UnitSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
```

5 Functions in scripts and chart expressions

```
Betacab|BB|5|10  
Betacab|CC|2|20  
Betacab|DD||25  
Canutility|AA|8|15  
Canutility|CC||19  
] (delimiter is '|');
```

sum

This script function returns the sum of expression over a number of records as defined by a **group by** clause.

Syntax:

```
sum ( [ distinct ] expression)
```

Arguments:

Argument	Description
distinct	If the word distinct occurs before the expression, all duplicates will be disregarded.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Example	Result	
Temp: LOAD * inline [Customer Product OrderNumber UnitsSales CustomerID Astrida AA 1 10 1 Astrida AA 7 18 1 Astrida BB 4 9 1 Astrida CC 6 2 1 Betacab AA 5 4 2 Betacab BB 2 5 2 Betacab DD Canutility DD 3 8 Canutility CC] (delimiter is ' ');	Customer	MySum
	A	39
	B	9
	C	8
Sum: LOAD SCustomer, Sum(UnitsSales) as MySum, Resident Temp Group By Customer;		

Sum - chart function

Sum() calculates the total of the values given by the expression or field across the aggregated data.


Syntax:

5 Functions in scripts and chart expressions

```
Sum ([{SetExpression}] [DISTINCT] [TOTAL [<fld {, fld}>]] expr)
```

Return data type: numeric

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded. <div style="border: 1px solid gray; padding: 10px; margin-top: 10px;"> <i>Although the DISTINCT qualifier is supported, use it only with extreme caution because it may mislead the reader into thinking a total value is shown when some data has been omitted.</i></div>
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Examples and results:

Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

5 Functions in scripts and chart expressions

Examples	Results
Sum(UnitsSales)	38. The total of the values in unitsSales.
Sum(UnitsSales*UnitPrice)	505. The total of unitPrice multiplied by unitsSales aggregated.
Sum (TOTAL UnitsSales*UnitPrice)	505 for all rows in the table as well as the total, because the TOTAL qualifier means the sum is still 505, disregarding the chart dimensions.
Make the selection Customer B. Sum({1} TOTAL UnitsSales*UnitPrice)	505, independent of the selection made, because the Set Analysis expression {1} defines the set of records to be evaluated as ALL, no matter what selection is made.

Data used in examples:

```
ProductData:
LOAD * inline [
Customer|Product|UnitsSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD||25
Canutility|AA|8|15
Canutility|CC||19
] (delimiter is '|');
```

Counter aggregation functions

Counter aggregation functions return various types of counts of an expression over a number of records in a data load script, or a number of values in a chart dimension.

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Counter aggregation functions in the data load script

count

Returns the count of expression over a number of records as defined by a **group by** clause.

```
count ([distinct ] expression | * )
```

MissingCount

Returns the missing count of expression over a number of records as defined by a **group by** clause.

```
MissingCount ([ distinct ] expression)
```

NullCount

Returns the NULL count of expression over a number of records as defined by a **group by** clause.

5 Functions in scripts and chart expressions

```
NullCount ([ distinct ] expression)
```

NumericCount

Returns the numeric count of expression over a number of records as defined by a **group by** clause.

```
NumericCount ([ distinct ] expression)
```

TextCount

Returns the text count of expression over a number of records as defined by a **group by** clause.

```
TextCount ([ distinct ] expression)
```

Counter aggregation functions in chart expressions

The following counter aggregation functions can be used in charts:



Chart aggregation functions can only be used on fields in chart expressions. The argument expression of one aggregation function must not contain another aggregation function.

Count

Count() is used to aggregate the number of values, text and numeric, in each chart dimension.

```
Count - chart function({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]  
expr)
```

MissingCount

MissingCount() is used to aggregate the number of missing values in each chart dimension. Missing values are all non-numeric values.

```
MissingCount - chart function({[SetExpression] [DISTINCT] [TOTAL [<fld  
{,fld}>]] expr)
```

NullCount

NullCount() is used to aggregate the number of NULL values in each chart dimension.

```
NullCount - chart function({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}  
>]]} expr)
```

NumericCount

NumericCount() aggregates the number of numeric values in each chart dimension.

```
NumericCount - chart function({[SetExpression] [DISTINCT] [TOTAL [<fld  
{,fld}>]]} expr)
```

TextCount

TextCount() is used to aggregate the number of field values that are non-numeric in each chart dimension.

```
TextCount - chart function({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}  
>]]} expr)
```

5 Functions in scripts and chart expressions

count

Returns the count of expression over a number of records as defined by a **group by** clause.

Syntax:

```
count ( [distinct ] expr | * )
```

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
distinct	If the word distinct occurs before the expression, all duplicates are disregarded.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Example	Result
<pre>Temp: LOAD * inline [Customer Product OrderNumber UnitsSales UnitPrice Astrida AA 1 4 16 Astrida AA 7 10 15 Astrida BB 4 9 9 Betacab CC 6 5 10 Betacab AA 5 2 20 Betacab BB 1 25 25 Canutility AA 3 8 15 Canutility CC 19 Divadip CC 2 4 16 Divadip DD 3 1 25] (delimiter is ' '); Count1: LOAD Customer,Count(OrderNumber) as OrdersByCustomer Resident Temp Group By Customer;</pre>	<pre>Customer OrdersByCustomer Astrida 3 Betacab 3 Canutility 2 Divadip 2</pre> <p>as long as the dimension Customer is included in the table on the sheet, otherwise the result for OrdersByCustomer is 3, 2.</p>
<p>Given that the Temp table is loaded as in the previous example:</p> <pre>LOAD Customer,Count(OrderNumber) as TotalOrdersNumber Resident Temp;</pre>	<pre>TotalOrderNumber 10</pre>

5 Functions in scripts and chart expressions

Example	Result
<p>Given that the Temp table is loaded as in the previous example:</p> <pre>LOAD Customer,Count(distinct OrderNumber) as TotalOrdersNumber Resident Temp;</pre>	<p>TotalOrderNumber 9 because there are two values of OrderNumber with the same value, 1.</p>

Count - chart function

Count() is used to aggregate the number of values, text and numeric, in each chart dimension.

Syntax:

```
Count ( {[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]] } expr )
```

Return data type: integer

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	<p>If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions.</p> <p>The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.</p>


Examples and results:

Customer	Product	OrderNumber	UnitSales	Unit Price
Astrida	AA	1	4	16
Astrida	AA	7	10	15
Astrida	BB	4	9	9
Betacab	BB	6	5	10

5 Functions in scripts and chart expressions

Customer	Product	OrderNumber	UnitSales	Unit Price
Betacab	CC	5	2	20
Betacab	DD	1	25	25
Canutility	AA	3	8	15
Canutility	CC			19
Divadip	AA	2	4	16
Divadip	DD	3		25

The following examples assume that all customers are selected, except where stated.

Example	Result
Count(OrderNumber)	10, because there are 10 fields that could have a value for OrderNumber, and all records, even empty ones, are counted. <div style="border: 1px solid gray; padding: 5px; margin: 5px 0;">  "0" counts as a value and not an empty cell. However if a measure aggregates to 0 for a dimension that dimension will not be included in charts. </div>
Count (Customer)	10, because Count evaluates the number of occurrences in all fields.
Count (DISTINCT [Customer])	4, because using the Distinct qualifier, Count only evaluates unique occurrences.
Given that customer Canutility is selected Count (OrderNumber)/Count ({1} TOTAL OrderNumber)	0.2, because the expression returns the number of orders from the selected customer as a percentage of orders from all customers. In this case 2 / 10.
Given that customers Astrida and Canutility are selected Count(TOTAL <Product> OrderNumber)	5, because that is the number of orders placed on products for the selected customers only and empty cells are counted.

Data used in examples:

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitsSales|UnitPrice
Astrida|AA|1|4|16
Astrida|AA|7|10|15
Astrida|BB|4|9|9
Betacab|CC|6|5|10
Betacab|AA|5|2|20
```

5 Functions in scripts and chart expressions

```
Betacab|BB|1|25| 25
Canutility|AA|3|8|15
Canutility|CC|||19
Divadip|CC|2|4|16
Divadip|DD|3|1|25
] (delimiter is '|');
```

MissingCount

Returns the missing count of expression over a number of records as defined by a **group by** clause.

Syntax:

```
MissingCount ( [ distinct ] expr)
```

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
distinct	If the word distinct occurs before the expression, all duplicates are disregarded.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Example	Result
<pre>Temp: LOAD * inline [Customer Product OrderNumber UnitsSales UnitPrice Astrida AA 1 4 16 Astrida AA 7 10 15 Astrida BB 4 9 9 Betacab CC 6 5 10 Betacab AA 5 2 20 Betacab BB 25 Canutility AA 15 Canutility CC 19 Divadip CC 2 4 16 Divadip DD 3 1 25] (delimiter is ' '); MissCount1: LOAD Customer,MissingCount(OrderNumber) as MissingOrdersByCustomer Resident Temp Group By Customer; Load MissingCount(OrderNumber2) as TotalMissingCount Resident Temp</pre>	<pre>Customer MissingOrdersByCustomer Astrida 0 Betacab 1 Canutility 2 Divadip 0 The second statement gives: TotalMissingCount 3 in a table with that dimension.</pre>

5 Functions in scripts and chart expressions

Example	Result
<p>Given that the Temp table is loaded as in the previous example:</p> <pre>LOAD Customer,MissingCount(distinct OrderNumber) as TotalMissingCountDistinct Resident Temp;</pre>	<p>TotalMissingCountDistinct 1 because there is only oneOrderNumber one missing value.</p>

MissingCount - chart function

MissingCount() is used to aggregate the number of missing values in each chart dimension. Missing values are all non-numeric values.

Syntax:

```
MissingCount ({ [SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]] expr)
```

Return data type: integer

Arguments:


Argument	Description
expr	The expression or field containing the data to be measured.
set_ expression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	<p>If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions.</p> <p>The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.</p>

Examples and results:

Customer	Product	OrderNumber	UnitSales	Unit Price
Astrida	AA	1	4	16
Astrida	AA	7	10	15
Astrida	BB	4	9	9
Betacab	BB	6	5	10

5 Functions in scripts and chart expressions

Customer	Product	OrderNumber	UnitSales	Unit Price
Betacab	CC	5	2	20
Betacab	DD			25
Canutility	AA			15
Canutility	CC			19
Divadip	AA	2	4	16
Divadip	DD	3		25

Example	Result
MissingCount ([OrderNumber])	3 because 3 of the 10 OrderNumber fields are empty <div style="border: 1px solid gray; padding: 5px; margin: 5px 0;">  "0" counts as a value and not an empty cell. However if a measure aggregates to 0 for a dimension that dimension will not be included in charts. </div>
MissingCount ([OrderNumber]) /MissingCount ({1} Total [OrderNumber])	The expression returns the number of incomplete orders from the selected customer as a fraction of incomplete orders from all customers. There is a total of 3 missing values for OrderNumber for all customers. So, for each Customer that has a missing value for Product the result is 1/3.

Data used in example:

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitsSales|UnitPrice
Astrida|AA|1|4|16
Astrida|AA|7|10|15
Astrida|BB|4|9|9
Betacab|CC|6|5|10
Betacab|AA|5|2|20
Betacab|BB||| 25
Canutility|AA|||15
Canutility|CC| |19
Divadip|CC|2|4|16
Divadip|DD|3|1|25
] (delimiter is '|');
```

NullCount

Returns the NULL count of expression over a number of records as defined by a **group by** clause.

Syntax:

```
NullCount ( [ distinct ] expr)
```

5 Functions in scripts and chart expressions

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
distinct	If the word distinct occurs before the expression, all duplicates are disregarded.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Example	Result
<pre>Set NULLINTERPRET = NULL; Temp: LOAD * inline [Customer Product OrderNumber UnitsSales CustomerID Astrida AA 1 10 1 Astrida AA 7 18 1 Astrida BB 4 9 1 Astrida CC 6 2 1 Betacab AA 5 4 2 Betacab BB 2 5 2 Betacab DD Canutility AA 3 8 Canutility CC NULL] (delimiter is ' '); Set NULLINTERPRET=; NullCount1: LOAD Customer,NullCount(OrderNumber) as NullOrdersByCustomer Resident Temp Group By Customer; LOAD NullCount(OrderNumber2) as TotalNullCount Resident Temp</pre>	<p>Customer NullOrdersByCustomer Astrida 0 Betacab 0 Canutility 1</p> <p>The second statement gives:</p> <p>TotalNullCount 1</p> <p>in a table with that dimension.</p>

NullCount - chart function

NullCount() is used to aggregate the number of NULL values in each chart dimension.

Syntax:

```
NullCount ({ [SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]] } expr)
```

Return data type: integer

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
expr	The expression or field containing the data to be measured.
set_ expression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Examples and results:

Example	Result
NullCount ([OrderNumber])	1 because we have introduced a null value using NullInterpret in the inline LOAD statement.

Data used in example:

```
Set NULLINTERPRET = NULL;
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitsSales|CustomerID
Astrida|AA|1|10|1
Astrida|AA|7|18|1
Astrida|BB|4|9|1
Astrida|CC|6|2|1
Betacab|AA|5|4|2
Betacab|BB|2|5|2
Betacab|DD|||
Canutility|AA|3|8|
Canutility|CC|NULL||
] (delimiter is '|');
Set NULLINTERPRET=;
```

NumericCount

Returns the numeric count of expression over a number of records as defined by a **group by** clause.

Syntax:

```
NumericCount ( [ distinct ] expr)
```

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
expr	The expression or field containing the data to be measured.
distinct	If the word distinct occurs before the expression, all duplicates are disregarded.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Example	Result
<pre>Temp: LOAD * inline [Customer Product OrderNumber UnitsSales UnitPrice Astrida AA 1 4 16 Astrida AA 7 10 15 Astrida BB 4 9 9 Betacab CC 6 5 10 Betacab AA 5 2 20 Betacab BB 25 Canutility AA 15 Canutility CC 19 Divadip CC 2 4 16 Divadip DD 3 1 25] (delimiter is ' '); NumCount1: LOAD Customer,NumericCount(OrderNumber) as NumericCountByCustomer Resident Temp Group By Customer; Load NumericCount(OrderNumber2) as TotalNumericCount Resident Temp</pre>	<p>Customer NumericCountByCustomer Astrida 0 Betacab 1 Canutility 2 Divadip 0</p> <p>The second statement gives:</p> <p>TotalNumericCount 3</p> <p>in a table with that dimension.</p>
<p>Given that the Temp table is loaded as in the previous example:</p> <pre>LOAD Customer,NumericCount(distinct OrderNumber) as TotalNumericCountDistinct Resident Temp;</pre>	<p>TotalNumericCountDistinct 1</p> <p>because there is only oneOrderNumber one missing value.</p>

NumericCount - chart function

NumericCount() aggregates the number of numeric values in each chart dimension.

Syntax:

```
NumericCount ({ [SetExpression] [DISTINCT] [TOTAL [<fld {, fld}>]] } expr)
```

Return data type: integer

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
expr	The expression or field containing the data to be measured.
set_expression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	<p>If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions.</p> <p>The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.</p>


Examples and results:

Customer	Product	OrderNumber	UnitSales	Unit Price
Astrida	AA	1	4	16
Astrida	AA	7	10	15
Astrida	BB	4	9	1
Betacab	BB	6	5	10
Betacab	CC	5	2	20
Betacab	DD			25
Canutility	AA			15
Canutility	CC			19
Divadip	AA	2	4	16
Divadip	DD	3		25

The following examples assume that all customers are selected, except where stated.

Example	Result
NumericCount ([OrderNumber])	7 because three of the 10 fields in OrderNumber are empty.

5 Functions in scripts and chart expressions

Example	Result
	 "0" counts as a value and not an empty cell. However if a measure aggregates to 0 for a dimension that dimension will not be included in charts.
NumericCount ([Product])	0 because all product names are in text. Typically you could use this to check that no text fields have been given numeric content.
NumericCount (DISTINCT [OrderNumber]) /Count(DISTINCT [OrderNumber])	Counts all the number of distinct numeric order numbers and divides it by the number of order numbers numeric and non-numeric. This will be 1 if all field values are numeric. Typically you could use this to check that all field values are numeric. In the example, there are 7 distinct numeric values for OrderNumber of 8 distinct numeric and non-numeric, so the expression returns 0.875.

Data used in example:

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitsSales|UnitPrice
Astrida|AA|1|4|16
Astrida|AA|7|10|15
Astrida|BB|4|9|9
Betacab|CC|6|5|10
Betacab|AA|5|2|20
Betacab|BB||| 25
Canutility|AA|||15
Canutility|CC| |19
Divadip|CC|2|4|16
Divadip|DD|3|1|25
] (delimiter is '|');
```

TextCount

Returns the text count of expression over a number of records as defined by a **group by** clause.

Syntax:

```
TextCount ( [ distinct ] expression)
```

Arguments:

Argument	Description
distinct	If the word distinct occurs before the expression, all duplicates are disregarded.

Example:

```
LOAD Month, TextCount(Item) as NumberOfTextItems from abc.csv group by Month;
```

TextCount - chart function

TextCount() is used to aggregate the number of field values that are non-numeric in each chart dimension.

Syntax:

```
TextCount ({[SetExpression] [DISTINCT] [TOTAL [<fld {, fld}>]]} expr)
```

Return data type: integer

Arguments:


Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Examples and results:

Customer	Product	OrderNumber	UnitSales	Unit Price
Astrida	AA	1	4	16
Astrida	AA	7	10	15
Astrida	BB	4	9	1
Betacab	BB	6	5	10
Betacab	CC	5	2	20
Betacab	DD			25
Canutility	AA			15
Canutility	CC			19

5 Functions in scripts and chart expressions

Customer	Product	OrderNumber	UnitSales	Unit Price
Divadip	AA	2	4	16
Divadip	DD	3		25

Example	Result
TextCount ([Product])	10 because all of the 10 fields in Product are text. <div style="border: 1px solid gray; padding: 5px;">  <i>"0" counts as a value and not an empty cell. However if a measure aggregates to 0 for a dimension that dimension will not be included in charts. Empty cells are evaluated as being non text and are not counted by TextCount.</i> </div>
TextCount ([OrderNumber])	3, because empty cells are counted. Typically, you would use this to check that no numeric fields have been given text values or are non-zero.
TextCount (DISTINCT [Product])/Count ([Product])	Counts all the number of distinct text values of Product (4), and divides it by the total number of values in Product (10). The result is 0.4.

Data used in example:

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitsSales|UnitPrice
Astrida|AA|1|4|16
Astrida|AA|7|1|15
Astrida|BB|4|9|9
Betacab|CC|6|5|10
Betacab|AA|5|2|20
Betacab|BB||| 25
Canutility|AA|||15
Canutility|CC|||19
Divadip|CC|2|4|16
Divadip|DD|3|1|25
] (delimiter is '|');
```

Financial aggregation functions

This section describes aggregation functions for financial operations regarding payments and cash flow.

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Financial aggregation functions in the data load script

IRR

5 Functions in scripts and chart expressions

This script function returns the aggregated internal rate of return for a series of cash flows represented by the numbers in expression iterated over a number of records as defined by a group by clause.

```
IRR (expression)
```

XIRR

This script function returns the aggregated internal rate of return for a schedule of cash flows (that is not necessarily periodic) represented by paired numbers in **valueexpression** and **dateexpression** iterated over a number of records as defined by a group by clause.

```
XIRR (valueexpression, dateexpression )
```

NPV

This script function returns the aggregated net present value of an investment based on a discount rate and a series of future payments (negative values) and incomes (positive values) represented by the numbers in expression iterated over a number of records as defined by a group by clause. The result has a default number format of money.

```
NPV (rate, expression)
```

XNPV

This script function returns the aggregated net present value for a schedule of cash flows (not necessarily periodic) represented by paired numbers in **valueexpression** and **dateexpression** iterated over a number of records as defined by a group by clause. Rate is the interest rate per period. The result has a default number format of **money**.

```
XNPV (rate, valueexpression, dateexpression)
```

Financial aggregation functions in chart expressions

These financial aggregation functions can be used in charts.

irr

IRR() returns the aggregated internal rate of return for a series of cash flows represented by the numbers in the expression given by **value** iterated over the chart dimensions.

```
IRR - chart function[TOTAL [<fld {,fld}>]] value)
```

npv

NPV() returns the aggregated net present value of an investment based on a **discount_rate** and a series of future payments (negative values) and incomes (positive values) represented by the numbers in **value** iterated over the chart dimensions. The result has a default number format of money. The payments and incomes are assumed to occur at the end of each period.

```
NPV - chart function([TOTAL [<fld {,fld}>]] discount_rate, value)
```

xirr

XIRR() returns the aggregated internal rate of return for a schedule of cash flows (that is not necessarily periodic) represented by paired numbers in the expressions given by **pmt** and **date** iterated over the chart

5 Functions in scripts and chart expressions

dimensions. All payments are discounted based on a 365-day year.

```
XIRR - chart function (page 180) ([TOTAL [<fld {,fld}>]] pmt, date)
```

xnpv

XNPV() returns the aggregated net present value for a schedule of cash flows (not necessarily periodic) represented by paired numbers in the expressions given by **pmt** and **date** iterated over the chart dimensions. The result has a default number format of money. All payments are discounted based on a 365-day year.

```
XNPV - chart function([TOTAL [<fld{,fld}>]] discount_rate, pmt, date)
```

IRR

This script function returns the aggregated internal rate of return for a series of cash flows represented by the numbers in expression iterated over a number of records as defined by a group by clause.

Syntax:

```
IRR(expression)
```

These cash flows do not have to be even, as they would be for an annuity. However, the cash flows must occur at regular intervals, such as monthly or annually. The internal rate of return is the interest rate received for an investment consisting of payments (negative values) and income (positive values) that occur at regular periods. The function needs at least one positive and one negative value to calculate.

Limitations:

Text values, NULL values and missing values are disregarded.

Example:

```
LOAD Year, IRR(Payments) as IRate from abc.csv  
group by Year;
```

IRR - chart function

IRR() returns the aggregated internal rate of return for a series of cash flows represented by the numbers in the expression given by **value** iterated over the chart dimensions.

These cash flows do not have to be even, as they would be for an annuity. However, the cash flows must occur at regular intervals, such as monthly or annually. The internal rate of return is the interest rate received for an investment consisting of payments (negative values) and income (positive values) that occur at regular periods. The function needs at least one positive and one negative value to calculate.

Syntax:

```
IRR([TOTAL [<fld {,fld}>]] value)
```

Return data type: numeric

Arguments:

5 Functions in scripts and chart expressions


Argument	Description
value	The expression or field containing the data to be measured.
TOTAL	<p>If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions.</p> <p>The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.</p>

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Text values, NULL values and missing values are disregarded.

Examples and results:

Example	Result
IRR (Payments)	<p>0.1634</p> <p>The payments are assumed to be periodic in nature, for example monthly.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <i>The Date field is used in the XIRR example where payments can be non-periodical as long as you provide the dates on which payments were made.</i></div>

Data used in examples::

```
Cashflow:
LOAD 2012 as Year, * inline [
Date,Discount,Payments
2012-01-01, 0.1,-10000
2012-03-01,0.1,3000
2012-10-30,0.1,4200
2013-02-01,0.1,6800];
```

See also:

- ▢ [XIRR - chart function \(page 180\)](#)
- ▢ [Aggr - chart function \(page 138\)](#)

NPV

This script function returns the aggregated net present value of an investment based on a discount rate and a series of future payments (negative values) and incomes (positive values) represented by the numbers in expression iterated over a number of records as defined by a group by clause. The result has a default number format of money.

Syntax:

```
NPV(rate, expression)
```

Rate is the interest rate per period. The payments and incomes are assumed to occur at the end of each period.

Limitations:

Text values, NULL values and missing values are disregarded.

Example:

```
LOAD Year, npv(0.05, Payments) as PValue from abc.csv group by Year;
```

NPV - chart function

NPV() returns the aggregated net present value of an investment based on a **discount_rate** and a series of future payments (negative values) and incomes (positive values) represented by the numbers in **value** iterated over the chart dimensions. The result has a default number format of money. The payments and incomes are assumed to occur at the end of each period.

Syntax:

```
NPV([TOTAL [<fld {,fld}>]] discount_rate, value)
```

Return data type: numeric

Arguments:

Argument	Description
discount_rate	discount_rate is the rate of discount over the length of the period.
value	The expression or field containing the data to be measured.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle

5 Functions in scripts and chart expressions

Argument	Description
	<p>brackets <fld>. These field names should be a subset of the chart dimension variables.</p> <p>The TOTAL qualifier may be followed by a list of one or more field names within angle brackets. These field names should be a subset of the chart dimension variables. In this case, the calculation is made disregarding all chart dimension variables except those listed, that is, one value is returned for each combination of field values in the listed dimension fields. Also, fields that are not currently a dimension in a chart may be included in the list. This may be useful in the case of group dimensions, where the dimension fields are not fixed. Listing all of the variables in the group causes the function to work when the drill-down level changes.</p>

Limitations:

discount_rate and **value** must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Text values, NULL values and missing values are disregarded.

Examples and results:

Example	Result
NPV(Discount, Payments)	1188.44

Data used in examples::

```
Cashflow:
LOAD 2012 as Year, * inline [
Date,Discount,Payments
2012-01-01, 0.1,-10000
2012-03-01,0.1,3000
2012-10-30,0.1,4200
2013-02-01,0.1,6800];
```

See also:

- [XNPV - chart function \(page 182\)](#)
- [Aggr - chart function \(page 138\)](#)

XIRR

This script function returns the aggregated internal rate of return for a schedule of cash flows (that is not necessarily periodic) represented by paired numbers in **valueexpression** and **dateexpression** iterated over a number of records as defined by a group by clause.

5 Functions in scripts and chart expressions

Syntax:

```
XIRR (valueexpression, dateexpression )
```

All payments are discounted based on a 365-day year.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair will result in the entire data-pair to be disregarded.

Example:

```
LOAD $ Year, XIRR(Payments, PayDates) as Irate from abc.csv group by Year;
```

XIRR - chart function

XIRR() returns the aggregated internal rate of return for a schedule of cash flows (that is not necessarily periodic) represented by paired numbers in the expressions given by **pmt** and **date** iterated over the chart dimensions. All payments are discounted based on a 365-day year.

Syntax:

```
XIRR ([TOTAL [<fld {,fld}>]] pmt, date)
```

Return data type: numeric

Arguments:

Argument	Description
pmt	Payments. The expression or field containing the cash flows corresponding to the payment schedule given in date .
date	The expression or field containing the schedule of dates corresponding to the cash flow payments given in pmt .
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

pmt and **date** must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

5 Functions in scripts and chart expressions

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Examples and results:

Example	Result
XIRR(Payments, Date)	0.5361

Data used in examples::

Cashflow:

```
LOAD 2012 as Year, * inline [
Date,Discount,Payments
2012-01-01, 0.1,-10000
2012-03-01,0.1,3000
2012-10-30,0.1,4200
2013-02-01,0.1,6800];
```

See also:

- ▢ [IRR - chart function \(page 176\)](#)
- ▢ [Aggr - chart function \(page 138\)](#)

XNPV

This script function returns the aggregated net present value for a schedule of cash flows (not necessarily periodic) represented by paired numbers in **valueexpression** and **dateexpression** iterated over a number of records as defined by a group by clause. Rate is the interest rate per period. The result has a default number format of **money**.

Syntax:

```
XNPV(rate, valueexpression, dateexpression)
```

All payments are discounted based on a 365-day year.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair will result in the entire data-pair to be disregarded.

Example:

```
LOAD Year, npv(0.05, Payments, PayDates) as PValue from abc.csv group by Year;
```

XNPV - chart function

XNPV() returns the aggregated net present value for a schedule of cash flows (not necessarily periodic) represented by paired numbers in the expressions given by **pmt** and **date** iterated over the chart dimensions. The result has a default number format of money. All payments are discounted based on a 365-day year.

Syntax:

```
XNPV ([TOTAL [<fld{,fld}>]] discount_rate, pmt, date)
```

Return data type: numeric

Arguments:

Argument	Description
discount_rate	discount_rate is the rate of discount over the length of the period.
pmt	Payments. The expression or field containing the cash flows corresponding to the payment schedule given in date .
date	The expression or field containing the schedule of dates corresponding to the cash flow payments given in pmt .
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

discount_rate, **pmt** and **date** must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** or **ALL** qualifiers. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Examples and results:

Example	Result
XNPV(Discount, Payments, Date)	2964.24 USD

Data used in examples::

Cashflow:

```
LOAD 2012 as Year, * inline [
Date,Discount,Payments
2012-01-01, 0.1,-10000
2012-03-01,0.1,3000
2012-10-30,0.1,4200
2013-02-01,0.1,6800];
```

See also:

- ▢ [NPV - chart function \(page 178\)](#)
- ▢ [Aggr - chart function \(page 138\)](#)

Statistical aggregation functions

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Statistical aggregation functions in the data load script

The following statistical aggregation functions can be used in scripts.

avg

This script function returns the average of expression over a number of records as defined by a **group by** clause.

```
avg ([distinct] expression)
```

correl

This script function returns the aggregated correlation coefficient for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
correl (x-expression, y-expression)
```

fractile

This script function returns the fractile of expression over a number of records as defined by a **group by** clause.

```
fractile (expression, fractile)
```

kurtosis

This script function returns the kurtosis of expression over a number of records as defined by a **group by** clause.

```
kurtosis ([distinct] expression )
```

linest_b

This script function returns the aggregated b value (y-intercept) of a linear regression defined by the equation

5 Functions in scripts and chart expressions

$y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
linest_b (y-expression, x-expression [, y0 [, x0 ]])
```

linest_df

This script function returns the aggregated degrees of freedom of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
linest_df (y-expression, x-expression [, y0 [, x0 ]])
```

linest_f

This script function returns the aggregated F statistic ($r^2/(1-r^2)$) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
linest_f (y-expression, x-expression [, y0 [, x0 ]])
```

linest_m

This script function returns the aggregated m value (slope) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
linest_m (y-expression, x-expression [, y0 [, x0 ]])
```

linest_r2

This script function returns the aggregated r^2 value (coefficient of determination) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
linest_r2 (y-expression, x-expression [, y0 [, x0 ]])
```

linest_seb

This script function returns the aggregated standard error of the b value of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
linest_seb (y-expression, x-expression [, y0 [, x0 ]])
```

linest_sem

This script function returns the aggregated standard error of the m value of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
linest_sem (y-expression, x-expression [, y0 [, x0 ]])
```

linest_sey

This script function returns the aggregated standard error of the y estimate of a linear regression defined by

5 Functions in scripts and chart expressions

the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
linest_sey (y-expression, x-expression [, y0 [, x0 ]])
```

linest_ssreg

This script function returns the aggregated regression sum of squares of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
linest_ssreg (y-expression, x-expression [, y0 [, x0 ]])
```

linest_ssresid

This script function returns the aggregated residual sum of squares of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
linest_ssresid (y-expression, x-expression [, y0 [, x0 ]])
```

median

This script function returns the aggregated median of expression over a number of records as defined by a **group by** clause.

```
median (expression)
```

skew

This script function returns the skewness of expression over a number of records as defined by a **group by** clause.

```
skew ([ distinct] expression)
```

stdev

This script function returns the standard deviation of expression over a number of records as defined by a **group by** clause.

```
stdev ([distinct] expression)
```

sterr

This script function returns the aggregated standard error ($stdev/\sqrt{n}$) for a series of values represented by expression iterated over a number of records as defined by a **group by** clause.

```
sterr ([distinct] expression)
```

steyx

This script function returns the aggregated standard error of the predicted y-value for each x-value in the regression for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
steyx (y-expression, x-expression)
```

Statistical aggregation functions in chart expressions

The following statistical aggregation functions can be used in charts.

avg

Avg() returns the aggregated average of the expression or field iterated over the chart dimensions.

```
Avg - chart function({[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]]} expr)
```

correl

Correl() returns the aggregated correlation coefficient for two data sets. The correlation function is a measure of the relationship between the data sets and is aggregated for (x,y) value pairs iterated over the chart dimensions.

```
Correl - chart function({[SetExpression] [TOTAL [<fld {, fld}>]]} value1, value2 )
```

fractile

Fractile() finds the value that corresponds to the fractile (quantile) of the aggregated data in the range given by the expression iterated over the chart dimensions.

```
Fractile - chart function({[SetExpression] [TOTAL [<fld {, fld}>]]} expr, fraction)
```

kurtosis

Kurtosis() finds the kurtosis of the range of data aggregated in the expression or field iterated over the chart dimensions.

```
Kurtosis - chart function({[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]]} expr)
```

linest_b

LINEST_B() returns the aggregated b value (y-intercept) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in the expressions given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_R2 - chart function({[SetExpression] [TOTAL [<fld{, fld}>]] }y_value, x_value[, y0_const[, x0_const]])
```

linest_df

LINEST_DF() returns the aggregated degrees of freedom of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in the expressions given by **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_DF - chart function({[SetExpression] [TOTAL [<fld{, fld}>]]} y_value, x_value [, y0_const [, x0_const]])
```

linest_f

5 Functions in scripts and chart expressions

LINEST_F() returns the aggregated F statistic ($r^2/(1-r^2)$) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in the expressions given by **x_value** and the **y_value**, iterated over the chart dimensions.

```
LINEST_F - chart function({[SetExpression] [TOTAL [<fld{, fld}>]]} y_value,  
x_value [, y0_const [, x0_const]])
```

linest_m

LINEST_M() returns the aggregated m value (slope) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_M - chart function({[SetExpression] [TOTAL [<fld{, fld}>]]} y_value,  
x_value [, y0_const [, x0_const]])
```

linest_r2

LINEST_R2() returns the aggregated r2 value (coefficient of determination) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_R2 - chart function({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_  
value, x_value[, y0_const[, x0_const]])
```

linest_seb

LINEST_SEB() returns the aggregated standard error of the b value of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_SEB - chart function({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_  
value, x_value[, y0_const[, x0_const]])
```

linest_sem

LINEST_SEM() returns the aggregated standard error of the m value of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_SEM - chart function({[set_expression]][ distinct ] [total [<fld  
{,fld}>] ] y-expression, x-expression [, y0 [, x0 ] ] )
```

linest_sey

LINEST_SEY() returns the aggregated standard error of the y estimate of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_SEY - chart function({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_  
value, x_value[, y0_const[, x0_const]])
```

linest_ssreg

5 Functions in scripts and chart expressions

LINEST_SSREG() returns the aggregated regression sum of squares of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_SSREG - chart function({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_value, x_value[, y0_const[, x0_const]])
```

linest_ssresid

LINEST_SSRESID() returns the aggregated residual sum of squares of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in the expressions given by **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_SSRESID - chart functionLINEST_SSRESID() returns the aggregated residual sum of squares of a linear regression defined by the equation  $y=mx+b$  for a series of coordinates represented by paired numbers in the expressions given by x_value and y_value, iterated over the chart dimensions. LINEST_SSRESID({[SetExpression]} [DISTINCT] [TOTAL [<fld{ ,fld}>]] y_value, x_value[, y0_const[, x0_const]])Return data type: numeric ArgumentDescriptiony_valueThe expression or field containing the range of y-values to be measured.x_valueThe expression or field containing the range of x-values to be measured.y0, x0An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do. SetExpressionBy default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression. DISTINCTIf the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded. TOTALIf the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. The expression must not contain aggregation functions, unless these inner aggregations contain the TOTAL qualifier. For more advanced nested aggregations, use the advanced aggregation function Aggr, in combination with calculated dimensions. Text values, NULL values and missing values in any or both pieces of a data-pair result in the
```

5 Functions in scripts and chart expressions

```
entire data-pair being disregarded. An example of how to use linest
functionsavg({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_value, x_value[,
y0_const[, x0_const]])
```

median

Median() returns the median value of the range of values aggregated in the expression iterated over the chart dimensions.

```
Median - chart function({[SetExpression] [TOTAL [<fld{ , fld}>]]) expr)
```

skew

Skew() returns the aggregated skewness of the expression or field iterated over the chart dimensions.

```
Skew - chart function{[SetExpression] [DISTINCT] [TOTAL [<fld{ ,fld}>]]}
expr)
```

stdev

Stdev() finds the standard deviation of the range of data aggregated in the expression or field iterated over the chart dimensions.

```
Stdev - chart function({[SetExpression] [DISTINCT] [TOTAL [<fld{ , fld}>]])
expr)
```

sterr

Sterr() finds the value of the standard error of the mean, (stdev/sqrt(n)), for the series of values aggregated in the expression iterated over the chart dimensions.

```
Sterr - chart function({[SetExpression] [DISTINCT] [TOTAL [<fld{ , fld}>]])
expr)
```

steyx

STEYX() returns the aggregated standard error when predicting y-values for each x-value in a linear regression given by a series of coordinates represented by paired numbers in the expressions given by **y_value** and **x_value**.

```
STEYX - chart function{[SetExpression] [TOTAL [<fld{ , fld}>]]} y_value, x_
value)
```

avg

This script function returns the average of expression over a number of records as defined by a **group by** clause.

Syntax:

```
avg ([distinct] expression)
```

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
distinct	If the word distinct occurs before the expression, all duplicates will be disregarded.

Example:

```
LOAD Month, avg(Sales) as AverageSalesPerMonth from abc.csv group by Month;
```

Avg - chart function

Avg() returns the aggregated average of the expression or field iterated over the chart dimensions.

Syntax:

```
Avg ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

Return data type: numeric

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Examples and results:

Customer	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Astrida	46	60	70	13	78	20	45	65	78	12	78	22

5 Functions in scripts and chart expressions

Customer	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Betacab	65	56	22	79	12	56	45	24	32	78	55	15
Canutility	77	68	34	91	24	68	57	36	44	90	67	27
Divadip	57	36	44	90	67	27	57	68	47	90	80	94

Customer	Sum([Sales])	Avg([Sales])	Avg(TOTAL Sales)	Avg(DISTINCT Sales)	Avg({1} TOTAL Sales)
	2566	53.46	53,458333	51,862069	53,458333
Astrida	587	48.92	53,458333	43,1	53,458333
Betacab	539	44.92	53,458333	43,909091	53,458333
Canutility	683	56.92	53,458333	55,909091	53,458333
Divadip	757	63.08	53,458333	61	53,458333

Example	Result
Avg(Sales)	For a table including the dimension customer and the measure Avg([Sales]), if Totals are shown, the result is 2566.
Avg([TOTAL Sales])	53.458333 for all values of customer, because the TOTAL qualifier means that dimensions are disregarded.
Avg(DISTINCT Sales)	51.862069 for the total, because using the Distinct qualifier means only unique values in sales for each customer are evaluated.

Data used in examples:

Monthnames:

```
LOAD * INLINE [
Month, Monthnumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];
```

Sales2013:

```
crosstable (Month, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

5 Functions in scripts and chart expressions

To get the months to sort in the correct order, when you create your visualizations, go to the **Sorting** section of the properties panel, select **Month** and mark the checkbox **Sort by expression**. In the expression box write Monthnumber.

See also:

- [Aggr - chart function \(page 138\)](#)

correl

This script function returns the aggregated correlation coefficient for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
correl (x-expression, y-expression)
```

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Example:

```
LOAD Month, correl(X,Y) as CC from abc.csv group by Month;
```

Correl - chart function

Correl() returns the aggregated correlation coefficient for two data sets. The correlation function is a measure of the relationship between the data sets and is aggregated for (x,y) value pairs iterated over the chart dimensions.

Syntax:

```
Correl ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] value1, value2 )
```

Return data type: numeric

Arguments:

Argument	Description
value1, value2	The expressions or fields containing the two sample sets for which the correlation coefficient is to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.

5 Functions in scripts and chart expressions

Argument	Description
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Examples and results:

Example	Result
correl (Age, salary)	For a table including the dimension Employee name and the measure correl(Age, salary), the result is 0.9270611. The result is only displayed for the totals cell.
correl (TOTAL Age, salary))	0.927. This and the following results are shown to three decimal places for readability. If you create a filter panel with the dimension Gender, and make selections from it, you see the result 0.951 when Female is selected and 0.939 if Male is selected. This is because the selection excludes all results that do not belong to the other value of Gender.
correl({1} TOTAL Age, salary))	0.927. Independent of selections. This is because the set expression {1} disregards all selections and dimensions.
correl (TOTAL <Gender> Age, salary))	0.927 in the total cell, 0.939 for all values of Male, and 0.951 for all values of Female. This corresponds to the results from making the selections in a filter panel based on Gender.

Data used in examples:

```
salary:
LOAD * inline [
"Employee name"|Gender|Age|salary
Aiden Charles|Male|20|25000
Brenda Davies|Male|25|32000
Charlotte Edberg|Female|45|56000
```

5 Functions in scripts and chart expressions

```
Daroush Ferrara|Male|31|29000
Eunice Goldblum|Female|31|32000
Freddy Halvorsen|Male|25|26000
Gauri Indu|Female|36|46000
Harry Jones|Male|38|40000
Ian Underwood|Male|40|45000
Jackie Kingsley|Female|23|28000
] (delimiter is '|');
```

See also:

- ▢ [Aggr - chart function \(page 138\)](#)
- ▢ [Avg - chart function \(page 190\)](#)
- ▢ [RangeCorrel \(page 479\)](#)

fractile

This script function returns the fractile of expression over a number of records as defined by a **group by** clause.

Syntax:

```
fractile(expression, fractile)
```

Example:

```
LOAD Class, fractile( Grade, 0.75 ) as F from abc.csv group by Class;
```

Fractile - chart function

Fractile() finds the value that corresponds to the fractile (quantile) of the aggregated data in the range given by the expression iterated over the chart dimensions.

Syntax:

```
Fractile([{{SetExpression}}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr,
fraction)
```

Return data type: numeric

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
fraction	A number between 0 and 1 corresponding to the fractile (quantile expressed as a fraction) to be calculated.

5 Functions in scripts and chart expressions

Argument	Description
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Examples and results:

Customer	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Astrida	46	60	70	13	78	20	45	65	78	12	78	22
Betacab	65	56	22	79	12	56	45	24	32	78	55	15
Canutility	77	68	34	91	24	68	57	36	44	90	67	27
Divadip	57	36	44	90	67	27	57	68	47	90	80	94

Example	Result
Fractile (Sales, 0.75)	For a table including the dimension customer and the measure Fractile([Sales]), if Totals are shown, the result is 71.75. This is the point in the distribution of values of sales that 75% of the values fall beneath.
Fractile (TOTAL Sales, 0.75))	71.75 for all values of customer, because the TOTAL qualifier means that dimensions are disregarded.
Fractile (DISTINCT Sales, 0.75)	70 for the total, because using the DISTINCT qualifier means only unique values in sales for each customer are evaluated.

Data used in examples:

5 Functions in scripts and chart expressions

```
Monthnames:
LOAD * INLINE [
Month, Monthnumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];
Sales2013:
```

```
crosstable (Month, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

To get the months to sort in the correct order, when you create your visualizations, go to the **Sorting** section of the properties panel, select **Month** and mark the checkbox **Sort by expression**. In the expression box write Monthnumber.

See also:

- [Aggr - chart function \(page 138\)](#)

kurtosis

This script function returns the kurtosis of expression over a number of records as defined by a **group by** clause.

Syntax:

```
kurtosis([distinct ] expression )
```

Arguments:

Argument	Description
distinct	If the word distinct occurs before the expression, all duplicates will be disregarded.

Example:

```
LOAD Month, kurtosis(Sales) as Saleskurtosis from abc.csv group by Month;
```

Kurtosis - chart function

Kurtosis() finds the kurtosis of the range of data aggregated in the expression or field iterated over the chart dimensions.

Syntax:

```
Kurtosis ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

Return data type: numeric

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Examples and results:

Type	Value																			
Comparison	2	2	3	3	1	1	1	3	3	1	2	3	2	1	2	1	3	2	3	2
		7	8	1		9		4										9	7	
Observation	35	4	1	1	2	1	4	1	2	4	1	3	3	4	3	2	1	3	1	2
		0	2	5	1	4	6	0	8	8	6	0	2	8	1	2	2	9	9	5

5 Functions in scripts and chart expressions

Example	Result
Kurtosis (value)	For a table including the dimension <code>type</code> and the measure <code>kurtosis(value)</code> , if Totals are shown for the table, and number formatting is set to 3 significant figures, the result is 1.252. For comparison it is 1.161 and for observation it is 1.115.
Kurtosis (TOTAL value)	1.252 for all values of <code>type</code> , because the TOTAL qualifier means that dimensions are disregarded.

Data used in examples:

Table1:

```
crosstable LOAD recno() as ID, * inline [
observation|comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

See also:

▢ [Avg - chart function \(page 190\)](#)

linest_b

This script function returns the aggregated b value (y-intercept) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
linest_b (y-expression, x-expression [, y0 [, x0 ]])
```

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Example:

```
LOAD Key, linest_b(Y,X) as Z from abc.csv group by Key;
```

LINEST_B - chart function


LINEST_B() returns the aggregated b value (y-intercept) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in the expressions given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:

```
LINEST_B ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value [, y0_const [, x0_const]])
```

Return data type: numeric

Arguments:

Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y0_const, x0_const	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate.  <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.

5 Functions in scripts and chart expressions

Argument	Description
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- ▢ *An example of how to use linest functions (page 227)*
- ▢ *Avg - chart function (page 190)*

linest_df

This script function returns the aggregated degrees of freedom of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
linest_df (y-expression, x-expression [, y0 [, x0 ]])
```

Arguments:

Argument	Description
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

5 Functions in scripts and chart expressions

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Example:

```
LOAD Key, linest_df(Y,X) as Z from abc.csv group by Key;
```

LINEST_DF - chart function


LINEST_DF() returns the aggregated degrees of freedom of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in the expressions given by **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:

```
LINEST_DF ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value [, y0_const [, x0_const]])
```

Return data type: numeric

Arguments:

Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. <div style="border: 1px solid gray; padding: 5px; margin-top: 10px;"> <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i></div>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

5 Functions in scripts and chart expressions

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- ▢ *An example of how to use linest functions (page 227)*
- ▢ *Avg - chart function (page 190)*

linest_f

This script function returns the aggregated F statistic ($r^2/(1-r^2)$) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
linest_f (y-expression, x-expression [, y0 [, x0 ]])
```

Arguments:

Argument	Description
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Example:

```
LOAD Key, linest_f(Y,X) as Z from abc.csv group by Key;
```

LINEST_F - chart function

LINEST_F() returns the aggregated F statistic ($r^2/(1-r^2)$) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in the expressions given by **x_value** and the **y_value**, iterated over the chart dimensions.


5 Functions in scripts and chart expressions

Syntax:

```
LINEST_F([SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value [, y0_const [, x0_const]])
```

Return data type: numeric

Arguments:

Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i></div>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- ▢ *An example of how to use linest functions (page 227)*

5 Functions in scripts and chart expressions

▢ *Avg - chart function (page 190)*

linest_m

This script function returns the aggregated m value (slope) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
linest_m (y-expression, x-expression [, y0 [, x0 ]])
```

Arguments:

Argument	Description
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Example:

```
LOAD Key, linest_m(Y,X) as Z from abc.csv group by Key;
```

LINEST_M - chart function

LINEST_M() returns the aggregated m value (slope) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:


```
LINEST_M([SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value [, y0_const [, x0_const]])
```

Return data type: numeric

Arguments:

Argument	Description
y_value	The expression or field containing the range of y-values to be measured.

5 Functions in scripts and chart expressions

Argument	Description
x_value	The expression or field containing the range of x-values to be measured.
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. <div style="border: 1px solid gray; padding: 5px; margin-top: 10px;"> <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i></div>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- ▢ [An example of how to use linest functions \(page 227\)](#)
- ▢ [Avg - chart function \(page 190\)](#)

linest_r2

This script function returns the aggregated r^2 value (coefficient of determination) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

5 Functions in scripts and chart expressions

```
linest_r2 (y-expression, x-expression [, y0 [, x0 ]])
```

Arguments:

Argument	Description
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Example:

```
LOAD Key, linest_r2(Y,X) as Z from abc.csv group by Key;
```

LINEST_R2 - chart function

LINEST_R2() returns the aggregated r2 value (coefficient of determination) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:


```
LINEST_R2([[SetExpression]] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value[, y0_const[, x0_const]])
```

Return data type: numeric

Arguments:

Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate.

5 Functions in scripts and chart expressions

Argument	Description
	 Unless both <i>y0</i> and <i>x0</i> are stated, the function requires at least two valid data-pairs to calculate. If <i>y0</i> and <i>x0</i> are stated, a single data pair will do.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- ▢ *An example of how to use linest functions (page 227)*
- ▢ *Avg - chart function (page 190)*

linest_seb

This script function returns the aggregated standard error of the b value of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
linest_seb (y-expression, x-expression [, y0 [, x0 ]])
```

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
y(0), x(0)	<p>An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate.</p> <p>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</p>

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Example:

```
LOAD Key, linest_seb(Y,X) as Z from abc.csv group by Key;
```

LINEST_SEB - chart function


LINEST_SEB() returns the aggregated standard error of the b value of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:

```
LINEST_SEB ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value[, y0_const[, x0_const]])
```

Return data type: numeric

Arguments:

Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y0, x0	<p>An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate.</p> <div style="border: 1px solid gray; padding: 10px; margin-top: 10px;">  <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i> </div>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.

5 Functions in scripts and chart expressions

Argument	Description
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- ▢ *An example of how to use linest functions (page 227)*
- ▢ *Avg - chart function (page 190)*

linest_sem

This script function returns the aggregated standard error of the m value of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
linest_sem (y-expression, x-expression [, y0 [, x0 ]])
```

Arguments:

Argument	Description
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

5 Functions in scripts and chart expressions

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Example:

```
LOAD Key, linest_sem(Y,X) as Z from abc.csv group by Key;
```

LINEST_SEM - chart function


LINEST_SEM() returns the aggregated standard error of the m value of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:

```
LINEST_SEM ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value[, y0_const[, x0_const]])
```

Return data type: numeric

Arguments:

Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. <div style="border: 1px solid gray; padding: 5px; margin-top: 10px;"> <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i></div>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

5 Functions in scripts and chart expressions

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- ▢ *An example of how to use `linest` functions (page 227)*
- ▢ *Avg - chart function (page 190)*

linest_sey

This script function returns the aggregated standard error of the y estimate of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
linest_sey (y-expression, x-expression [, y0 [, x0 ]])
```

Arguments:

Argument	Description
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Example:

```
LOAD Key, linest_sey(Y,X) as Z from abc.csv group by Key;
```

LINEST_SEY - chart function

LINEST_SEY() returns the aggregated standard error of the y estimate of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.


5 Functions in scripts and chart expressions

Syntax:

```
LINEST_SEY ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value[, y0_const[, x0_const]])
```

Return data type: numeric

Arguments:

Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i></div>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- ▢ *An example of how to use linest functions (page 227)*

▢ *Avg - chart function (page 190)*

linest_ssreg

This script function returns the aggregated regression sum of squares of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
linest_ssreg (y-expression, x-expression [, y0 [, x0 ]])
```

Arguments:

Argument	Description
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Example:

```
LOAD Key, linest_ssreg(Y,X) as Z from abc.csv group by Key;
```

LINEST_SSREG - chart function

LINEST_SSREG() returns the aggregated regression sum of squares of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:


```
LINEST_SSREG ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value,  
x_value[, y0_const[, x0_const])
```

Return data type: numeric

Arguments:

Argument	Description
y_value	The expression or field containing the range of y-values to be measured.

5 Functions in scripts and chart expressions

Argument	Description
x_value	The expression or field containing the range of x-values to be measured.
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i></div>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- ▢ [An example of how to use linest functions \(page 227\)](#)
- ▢ [Avg - chart function \(page 190\)](#)

linest_ssresid

This script function returns the aggregated residual sum of squares of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

5 Functions in scripts and chart expressions

```
linest_ssresid (y-expression, x-expression [, y0 [, x0 ]])
```

Arguments:

Argument	Description
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Example:

```
LOAD Key, linest_ssresid(Y,X) as Z from abc.csv group by Key;
```

LINEST_SSRESID - chart function

LINEST_SSRESID() returns the aggregated residual sum of squares of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in the expressions given by **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:


```
LINEST_SSRESID ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value[, y0_const[, x0_const]])
```

Return data type: numeric

Arguments:

Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate.

5 Functions in scripts and chart expressions

Argument	Description
	 Unless both <i>y0</i> and <i>x0</i> are stated, the function requires at least two valid data-pairs to calculate. If <i>y0</i> and <i>x0</i> are stated, a single data pair will do.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

An optional value *y0* may be stated forcing the regression line to pass through the y-axis at a given point. By stating both *y0* and *x0* it is possible to force the regression line to pass through a single fixed coordinate.

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- ▢ *An example of how to use linest functions (page 227)*
- ▢ *Avg - chart function (page 190)*

median

This script function returns the aggregated median of expression over a number of records as defined by a **group by** clause.

Syntax:

```
median (expression)
```

Example:

```
LOAD Class, Median(Grade) as MG from abc.csv group by Class;
```


Median - chart function

Median() returns the median value of the range of values aggregated in the expression iterated over the chart dimensions.

Syntax:

```
Median ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

Return data type: numeric

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Examples and results:

Type	Value																			
Comparison	2	2	3	3	1	1	1	3	3	1	2	3	2	1	2	1	3	2	3	2
		7	8	1		9		4										9	7	
Observation	35	4	1	1	2	1	4	1	2	4	1	3	3	4	3	2	1	3	1	2
		0	2	5	1	4	6	0	8	8	6	0	2	8	1	2	2	9	9	5

5 Functions in scripts and chart expressions

Example	Result
Median (value)	For a table including the dimension type and the measure Median(value), if Totals are shown, the result is 19, for comparison it is 2.5 and for observation it is 26.5.
Median (TOTAL value))	19 for all values of type, because the TOTAL qualifier means that dimensions are disregarded.

Data used in examples:

Table1:

```
crosstable LOAD recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

See also:

- [Avg - chart function \(page 190\)](#)

skew

This script function returns the skewness of expression over a number of records as defined by a **group by** clause.

Syntax:

```
skew([ distinct] expression)
```

Example:

```
LOAD Month, skew(Sales) as SalesSkew from abc.csv group by Month;
```

5 Functions in scripts and chart expressions

Skew - chart function

Skew() returns the aggregated skewness of the expression or field iterated over the chart dimensions.

Syntax:

```
Skew ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Examples and results:

Type	Value																			
Comparison	2	2	3	3	1	1	1	3	3	1	2	3	2	1	2	1	3	2	3	2
		7	8	1		9		4										9	7	
Observation	35	4	1	1	2	1	4	1	2	4	1	3	3	4	3	2	1	3	1	2
		0	2	5	1	4	6	0	8	8	6	0	2	8	1	2	2	9	9	5

Example	Result
Skew	For a table including the dimension type and the measure skew(value), if Totals are shown,

5 Functions in scripts and chart expressions

Example	Result
(value)	and number formatting is set to 3 significant figures, the result is 0.235. For comparison it is 0.864, and for observation 0.3265.
Skew (TOTAL value))	0.235 for all values of type, because the TOTAL qualifier means that dimensions are disregarded.

Data used in examples:

Table1:

```
crosstable LOAD recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

See also:

▢ [Avg - chart function \(page 190\)](#)

stdev

This script function returns the standard deviation of expression over a number of records as defined by a **group by** clause.

Syntax:

```
stdev ([distinct] expression)
```

Arguments:

Argument	Description
distinct	If the word distinct occurs before the expression, all duplicates will be disregarded.

5 Functions in scripts and chart expressions

LOAD Month, stdev(Sales) as SalesStandardDeviation from abc.csv group by Month;

Stdev - chart function

Stdev() finds the standard deviation of the range of data aggregated in the expression or field iterated over the chart dimensions.

Syntax:

```
Stdev ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

Return data type: numeric

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Examples and results:

Type	Value																			
Comparison	2	2	3	3	1	1	1	3	3	1	2	3	2	1	2	1	3	2	3	2
		7	8	1		9		4										9	7	
Observation	35	4	1	1	2	1	4	1	2	4	1	3	3	4	3	2	1	3	1	2
		0	2	5	1	4	6	0	8	8	6	0	2	8	1	2	2	9	9	5

5 Functions in scripts and chart expressions

Example	Result
stdev (value)	For a table including the dimension type and the measure stdev(value), if Totals are shown, the result is 15.475, for comparison it is 14.612 and for observation it is 12.508.
stdev (TOTAL value))	is 15.475 for all values of type, because the TOTAL qualifier means that dimensions are disregarded.

Data used in examples:

Table1:

```
crosstable LOAD recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

See also:

- ▢ [Avg - chart function \(page 190\)](#)
- ▢ [STEYX - chart function \(page 225\)](#)

sterr

This script function returns the aggregated standard error (stdev/\sqrt{n}) for a series of values represented by expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
sterr ([distinct] expression)
```

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
distinct	If the word distinct occurs before the expression, all duplicates will be disregarded.

Limitations:

Text values, NULL values and missing values are disregarded.

Example:

```
LOAD Key, sterr(X) as Z from abc.csv group by Key;
```

Sterr - chart function

Sterr() finds the value of the standard error of the mean, (stdev/sqrt(n)), for the series of values aggregated in the expression iterated over the chart dimensions.

Syntax:

```
Sterr ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

Return data type: numeric

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Text values, NULL values and missing values are disregarded.

5 Functions in scripts and chart expressions

Examples and results:

Type	Value																			
Comparison	2	2	3	3	1	1	1	3	3	1	2	3	2	1	2	1	3	2	3	2
		7	8	1		9		4										9	7	
Observation	35	4	1	1	2	1	4	1	2	4	1	3	3	4	3	2	1	3	1	2
		0	2	5	1	4	6	0	8	8	6	0	2	8	1	2	2	9	9	5

Example	Result
sterr (value)	For a table including the dimension type and the measure sterr(value), if Totals are shown, the result is 2.447, for comparison it is 3.267 and for observation it is 2.797.
sterr (TOTAL value))	2.447 for all values of type, because the TOTAL qualifier means that dimensions are disregarded.

Data used in examples:

```
Table1:
crosstable LOAD recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

See also:

- ▢ [Avg - chart function \(page 190\)](#)
- ▢ [STEYX - chart function \(page 225\)](#)

steyx

This script function returns the aggregated standard error of the predicted y-value for each x-value in the regression for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
steyx (y-expression, x-expression)
```

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Example:

```
LOAD Key, steyx(Y,X) as Z from abc.csv group by Key;
```

STEYX - chart function

STEYX() returns the aggregated standard error when predicting y-values for each x-value in a linear regression given by a series of coordinates represented by paired numbers in the expressions given by **y_value** and **x_value**.

Syntax:

```
STEYX ([[SetExpression]] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value)
```

Return data type: numeric

Arguments:

Argument	Description
y_value	The expression or field containing the range of known y-values to be measured.
x_value	The expression or field containing the range of known x-values to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

5 Functions in scripts and chart expressions

Limitations:

The expression must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced aggregation function **Aggr**, in combination with calculated dimensions.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Examples and results:

Data series												
KnownX	17	16	14	11	10	8	7	6	5	5	5	4
KnownY	15	14	12	9	9	10	6	2	3	5	8	7

Example	Result
Steyx(KnownY, KnownX)	2.071 (If number formatting is set to 3 decimal places.)
Steyx(TOTAL KnownY, KnownX))	2.071 across all dimensions, if no selections are made. 2.121 across all dimensions, if the selections, 4, 5, and 6 are made for KnownX, for example.

Data used in examples:

```
Trend:
LOAD * inline [
Month, KnownY, KnownX
Jan, 2, 6
Feb, 3, 5
Mar, 9, 11
Apr, 6, 7
May, 8, 5
Jun, 7, 4
Jul, 5, 5
Aug, 10, 8
Sep, 9, 10
Oct, 12, 14
Nov, 15, 17
Dec, 14, 16
] (delimiter is ',';
```

See also:

- ▢ [Avg - chart function \(page 190\)](#)
- ▢ [Sterr - chart function \(page 223\)](#)

An example of how to use linest functions


The linest functions are used to find values associated with linear regression analysis. This section describes how to build visualizations using sample data to find the values of the linest functions available in Qlik Sense. Please refer to the individual linest chart function topics for descriptions of syntax and arguments.

Loading the sample data

Do the following:



1. Create a new app.
2. In the data load editor, enter the following:

```
T1:
LOAD *, 1 as Grp;
LOAD * inline [
X |Y
1| 0
2|1
3|3
4| 8
5| 14
6| 20
7| 0
8| 50
9| 25
10| 60
11| 38
12| 19
13| 26
14| 143
15| 98
16| 27
17| 59
18| 78
19| 158
20| 279 ] (delimiter is '|');
R1:
LOAD
Grp,
linest_B(Y,X) as Linest_B,
linest_DF(Y,X) as Linest_DF,
linest_F(Y,X) as Linest_F,
linest_M(Y,X) as Linest_M,
linest_R2(Y,X) as Linest_R2,
linest_SEB(Y,X,1,1) as Linest_SEB,
linest_SEM(Y,X) as Linest_SEM,
linest_SEY(Y,X) as Linest_SEY,
linest_SSREG(Y,X) as Linest_SSREG,
linest_SSRESID(Y,X) as Linest_SSRESID
resident T1 group by Grp;
```

3. Click  to load the data.

5 Functions in scripts and chart expressions

Creating the linest chart function visualizations

1. In the data load editor, click  to go to the app view, create a new sheet and open it..
2. Click  **Edit** to edit the sheet.
3. From **Charts** add a line chart, and from **Fields** add X as a dimension and Sum(Y) as a measure.
A line chart is create that represents the graph of X plotted against Y, from which the linest functions are calculated.
4. From **Charts** add a table with the following expressions as a dimension:
valueList('Linest_b', 'Linest_df','Linest_f', 'Linest_m','Linest_r2','Linest_SEB','Linest_SEM','Linest_SEY','Linest_SSREG','Linest_SSRESID')
This uses the synthetic dimensions function to create labels for the dimensions with the names of the linest functions. You can change the label to **Linest functions** to save space.
5. Add the following expression to the table as a measure:
Pick(Match(ValueList('Linest_b', 'Linest_df','Linest_f', 'Linest_m','Linest_r2','Linest_SEB','Linest_SEM','Linest_SEY','Linest_SSREG','Linest_SSRESID'),'Linest_b', 'Linest_df','Linest_f', 'Linest_m','Linest_r2','Linest_SEB','Linest_SEM','Linest_SEY','Linest_SSREG','Linest_SSRESID'),Linest_b(Y,X),Linest_df(Y,X),Linest_f(Y,X),Linest_m(Y,X),Linest_r2(Y,X),Linest_SEB(Y,X,1,1),Linest_SEM(Y,X),Linest_SEY(Y,X),Linest_SSREG(Y,X),Linest_SSRESID(Y,X))
This displays the value of the result of each linest function against the corresponding name in the synthetic dimension. The result of Linest_b(Y,X) is displayed next to **linest_b**, and so on.

Result

Linest functions	Linest function results
Linest_b	-35.047
Linest_df	18
Linest_f	20.788
Linest_m	8.605
Linest_r2	0.536
Linest_SEB	22.607
Linest_SEM	1.887
Linest_SEY	48.666
Linest_SSREG	49235.014
Linest_SSRESID	42631.186

Tables for the sample data containing measure using the linest functions would look like this:

Linest_B	Linest_DF	Linest_F	Linest_M	Linest_R2	Linest_SEB
-35.047	18	20.788	8.605	0.536	22.607

Linest_SEM	Linest_SEY	Linest_SSREG	Linest_SSRESID
1.887	48.666	49235.014	42631.186

Statistical test functions

This section describes functions for statistical tests, which are divided into three categories. The functions can be used in both the data load script and chart expressions, but the syntax differs.

Chi-2 test functions

Generally used in the study of qualitative variables. One can compare observed frequencies in a one-way frequency table with expected frequencies, or study the connection between two variables in a contingency table.

T-test functions

A statistical examination of two population means. A two-sample t-test examines whether two samples are different and is commonly used when two normal distributions have **unknown variances** and when an experiment uses a **small sample size**.

Z-test functions

A statistical examination of two population means. A two sample z-test examines whether two samples are different and is commonly used when two normal distributions have **known variances** and when an experiment uses a **large sample size**.

Chi2-test functions

Generally used in the study of qualitative variables. One can compare observed frequencies in a one-way frequency table with expected frequencies, or study the connection between two variables in a contingency table.

Chi2-test functions in the data load script

Chi2Test_p

This script function returns the aggregated chi2-test p value (significance) for one or two series of values iterated over a number of records as defined by a **group by** clause.

```
Chi2Test_p (col, row, observed_value [, expected_value])
```

Chi2Test_df

This script function returns the aggregated chi2-test df value (degrees of freedom) for one or two series of values iterated over a number of records as defined by a **group by** clause.

```
Chi2Test_df (col, row, observed_value [, expected_value])
```

Chi2Test_chi2

This Script function returns the aggregated chi2-test value for one or two series of values iterated over a

5 Functions in scripts and chart expressions

number of records as defined by a **group by** clause.

```
Chi2Test_chi2 (col, row, observed_value [, expected_value])
```

Chi2-test functions in charts

Chi2Test_chi2

Chi2Test_chi2() returns the aggregated chi²-test value for one or two series of values iterated over the chart dimensions.

```
Chi2Test_chi2 - chart function(col, row, actual_value[, expected_value])
```

chi2test_df

Chi2Test_df() returns the aggregated chi²-test df value (degrees of freedom) for one or two series of values iterated over the chart dimensions.

```
Chi2Test_df - chart function(col, row, actual_value[, expected_value])
```

Chi2test_p

Chi2Test_p() returns the aggregated chi²-test p value (significance) for one or two series of values iterated over the chart dimensions.

```
Chi2Test_p - chart function(col, row, actual_value[, expected_value])
```

See also:

- ▢ *T-test functions in charts (page 234)*
- ▢ *Z-test functions in charts (page 289)*

Chi2Test_chi2

This Script function returns the aggregated chi²-test value for one or two series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
Chi2Test_chi2 (col, row, observed_value [, expected_value])
```

Arguments:

Argument	Description
col	The specified column in the matrix of values being tested.
row	The specified row in the matrix of values being tested.
observed_value	The observed value of the data at the specified col and row .
expected_value	The expected value for the distribution at the specified col and row .

Example:

5 Functions in scripts and chart expressions

```
LOAD Year, Chi2Test_chi2(Gender,Description,Observed,Expected) as X from abc.csv group by Year;
```

Chi2Test_chi2 - chart function

Chi2Test_chi2() returns the aggregated chi²-test value for one or two series of values iterated over the chart dimensions.



All Qlik Sense chi²-test functions have the same arguments.

Syntax:

```
Chi2Test_chi2(col, row, actual_value[, expected_value])
```

Return data type: numeric

Arguments:

Argument	Description
col, row	The specified column and row in the matrix of values being tested..
actual_value	The observed value of the data at the specified col and row .
expected_value	The expected value for the distribution at the specified col and row .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
Chi2Test_chi2( Grp, Grade, Count )  
Chi2Test_chi2( Gender, Description, Observed, Expected )
```

See also:

▫ [Examples of how to use chi2-test functions \(page 310\)](#)

Chi2Test_df

This script function returns the aggregated chi2-test df value (degrees of freedom) for one or two series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
Chi2Test_df (col, row, observed_value [, expected_value])
```

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
col	The specified column in the matrix of values being tested.
row	The specified row in the matrix of values being tested.
observed_value	The observed value of the data at the specified col and row .
expected_value	The expected value for the distribution at the specified col and row .

Example:

```
LOAD Year, Chi2Test_df(Gender,Description,Observed,Expected) as X from abc.csv group by Year;
```

Chi2Test_df - chart function

Chi2Test_df() returns the aggregated chi²-test df value (degrees of freedom) for one or two series of values iterated over the chart dimensions.



All Qlik Sense chi²-test functions have the same arguments.

Syntax:

```
Chi2Test_df(col, row, actual_value[, expected_value])
```

Return data type: numeric

Arguments:

Argument	Description
col, row	The specified column and row in the matrix of values being tested.
actual_value	The observed value of the data at the specified col and row .
expected_value	The expected value for the distribution at the specified col and row .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
Chi2Test_df( Grp, Grade, Count )  
Chi2Test_df( Gender, Description, Observed, Expected )
```

See also:

▢ [Examples of how to use chi²-test functions \(page 310\)](#)

Chi2Test_p - chart function

Chi2Test_p() returns the aggregated chi²-test p value (significance) for one or two series of values iterated over the chart dimensions. The test can be done either on the values in **actual_value**, testing for variations within the specified **col** and **row** matrix, or by comparing values in **actual_value** with corresponding values in **expected_value**, if specified.



All Qlik Sense chi²-test functions have the same arguments.

Syntax:

```
Chi2Test_p(col, row, actual_value[, expected_value])
```

Return data type: numeric

Arguments:

Argument	Description
col, row	The specified column and row in the matrix of values being tested.
actual_value	The observed value of the data at the specified col and row .
expected_value	The expected value for the distribution at the specified col and row .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
Chi2Test_p( Grp, Grade, Count )  
Chi2Test_p( Gender, Description, Observed, Expected )
```

See also:

- ▢ [Examples of how to use chi²-test functions \(page 310\)](#)

Chi2Test_p

This script function returns the aggregated chi²-test p value (significance) for one or two series of values iterated over a number of records as defined by a **group by** clause.

The test can be done either on the values in **observed_value** testing for variations within the specified **col** and **row** matrix or by comparing values in **observed_value** with corresponding values in **expected_values**. Text values, NULL values and missing values in the value expressions will result in the function returning NULL.

Syntax:

```
Chi2Test_p (col, row, observed_value [, expected_value])
```

Arguments:

Argument	Description
col	The specified column in the matrix of values being tested.
row	The specified row in the matrix of values being tested.
observed_value	The observed value of the data at the specified col and row .
expected_value	The expected value for the distribution at the specified col and row .

Example:

```
LOAD Year, Chi2Test_p(Gender,Description,Observed,Expected) as X from abc.csv group by Year
```

T-test functions in charts

A statistical examination of two population means. A two-sample t-test examines whether two samples are different and is commonly used when two normal distributions have **unknown variances** and when an experiment uses a **small sample size**.

In the following sections, the t-test statistical test functions are grouped according to the sample student test that applies to each type of function.

T-test functions in the data load script

Two independent samples t-tests

The following eight functions apply to two independent samples student's t-tests.

TTest_t

This script function returns the aggregated t value for two independent series of values iterated over a number of records as defined by a **group by** clause.

```
TTest_t (group, value [, eq_var = true])
```

TTest_df

This script function returns the aggregated student's t-test df value (degrees of freedom) for two independent series of values iterated over a number of records as defined by a **group by** clause.

```
TTest_df (group, value [, eq_var = true])
```

TTest_sig

This script function returns the aggregated student's t-test 2-tailed level of significance for two independent series of values iterated over a number of records as defined by a **group by** clause.

5 Functions in scripts and chart expressions

```
TTest_sig (group, value [, eq_var = true])
```

TTest_dif

This script function returns the aggregated student's t-test mean difference for two independent series of values iterated over a number of records as defined by a **group by** clause.

```
TTest_dif (group, value [, eq_var = true])
```

TTest_sterr

This script function returns the aggregated student's t-test standard error of the mean difference for two independent series of values iterated over a number of records as defined by a **group by** clause.

```
TTest_sterr (group, value [, eq_var = true])
```

TTest_conf

This script function returns the aggregated t value for two independent series of values iterated over a number of records as defined by a **group by** clause.

```
TTest_conf (group, value [, sig = 0.025 [, eq_var = true]])
```

TTest_lower

This script function returns the aggregated value for the lower end of the confidence interval for two independent series of values iterated over a number of records as defined by a **group by** clause.

```
TTest_lower (group, value [, sig = 0.025 [, eq_var = true]])
```

TTest_upper

This script function returns the aggregated value for the upper end of the confidence interval for two independent series of values iterated over a number of records as defined by a **group by** clause.

```
TTest_upper (group, value [, sig = 0.025 [, eq_var = true]])
```

Two independent weighted samples t-tests

The following eight functions apply to two independent samples student's t-tests where the input data series is given in weighted two-column format.

TTestw_t

This script function returns the aggregated t value for two independent series of values iterated over a number of records as defined by a **group by** clause.

```
TTestw_t (weight, group, value [, eq_var = true])
```

TTestw_df

This script function returns the aggregated student's t-test df value (degrees of freedom) for two independent series of values iterated over a number of records as defined by a **group by** clause.

```
TTestw_df (weight, group, value [, eq_var = true])
```

TTestw_sig

5 Functions in scripts and chart expressions

This script function returns the aggregated student's t-test 2-tailed level of significance for two independent series of values iterated over a number of records as defined by a **group by** clause.

```
TTestw_sig (weight, group, value [, eq_var = true])
```

TTestw_dif

This script function returns the aggregated student's t-test mean difference for two independent series of values iterated over a number of records as defined by a **group by** clause.

```
TTestw_dif (weight, group, value [, eq_var = true])
```

TTestw_sterr

This script function returns the aggregated student's t-test standard error of the mean difference for two independent series of values iterated over a number of records as defined by a **group by** clause.

```
TTestw_sterr (weight, group, value [, eq_var = true])
```

TTestw_conf

This script function returns the aggregated t value for two independent series of values iterated over a number of records as defined by a **group by** clause.

```
TTestw_conf (weight, group, value [, sig = 0.025 [, eq_var = true]])
```

TTestw_lower

This script function returns the aggregated value for the lower end of the confidence interval for two independent series of values iterated over a number of records as defined by a **group by** clause.

```
TTestw_lower (weight, group, value [, sig = 0.025 [, eq_var = true]])
```

TTestw_upper

This script function returns the aggregated value for the upper end of the confidence interval for two independent series of values iterated over a number of records as defined by a **group by** clause.

```
TTestw_upper (weight, group, value [, sig = 0.025 [, eq_var = true]])
```

One sample t-tests

The following eight functions apply to one-sample student's t-tests.

TTest1_t

This script function returns the aggregated t value for a series of values iterated over a number of records as defined by a **group by** clause.

```
TTest1_t (value)
```

TTest1_df

This script function returns the aggregated student's t-test df value (degrees of freedom) for a series of values iterated over a number of records as defined by a **group by** clause.

```
TTest1_df (value)
```

TTest1_sig

5 Functions in scripts and chart expressions

This script function returns the aggregated student's t-test 2-tailed level of significance for a series of values iterated over a number of records as defined by a **group by** clause.

```
TTest1_sig (value)
```

TTest1_dif

This script function returns the aggregated student's t-test mean difference for a series of values iterated over a number of records as defined by a **group by** clause.

```
TTest1_dif (value)
```

TTest1_sterr

This script function returns the aggregated student's t-test standard error of the mean difference for a series of values iterated over a number of records as defined by a **group by** clause.

```
TTest1_sterr (value)
```

TTest1_conf

This script function returns the aggregated t value for a series of values iterated over a number of records as defined by a **group by** clause.

```
TTest1_conf (value [, sig = 0.025 ])
```

TTest1_lower

This script function returns the aggregated value for the lower end of the confidence interval for a series of values iterated over a number of records as defined by a **group by** clause.

```
TTest1_lower (value [, sig = 0.025 ])
```

TTest1_upper

This script function returns the aggregated value for the upper end of the confidence interval for a series of values iterated over a number of records as defined by a **group by** clause.

```
TTest1_upper (value [, sig = 0.025 ])
```

One weighted sample t-tests

The following eight functions apply to one-sample student's t-tests where the input data series is given in weighted two-column format.

TTest1w_t

This script function returns the aggregated t value for a series of values iterated over a number of records as defined by a **group by** clause.

```
TTest1w_t (weight, value)
```

TTest1w_df

This script function returns the aggregated student's t-test df value (degrees of freedom) for a series of values iterated over a number of records as defined by a **group by** clause.

```
TTest1w_df (weight, value)
```

5 Functions in scripts and chart expressions

TTestw_sig

This script function returns the aggregated student's t-test 2-tailed level of significance for a series of values iterated over a number of records as defined by a **group by** clause.

```
TTest1w_sig (weight, value)
```

TTest1w_dif

This script function returns the aggregated student's t-test mean difference for a series of values iterated over a number of records as defined by a **group by** clause.

```
TTest1w_dif (weight, value)
```

TTest1w_sterr

This script function returns the aggregated student's t-test standard error of the mean difference for a series of values iterated over a number of records as defined by a **group by** clause.

```
TTest1w_sterr (weight, value)
```

TTest1w_conf

This script function returns the aggregated t value for a series of values iterated over a number of records as defined by a **group by** clause.

```
TTest1w_conf (weight, value [, sig = 0.025 ])
```

TTest1w_lower

This script function returns the aggregated value for the lower end of the confidence interval for a series of values iterated over a number of records as defined by a **group by** clause.

```
TTest1w_lower (weight, value [, sig = 0.025 ])
```

TTest1w_upper

This script function returns the aggregated value for the upper end of the confidence interval for a series of values iterated over a number of records as defined by a **group by** clause.

```
TTest1w_upper (weight, value [, sig = 0.025 ])
```

T-test functions in chart expressions

Example:

- *Creating a typical t-test report (page 312)*

Two independent samples t-tests

The following functions apply to two independent samples student's t-tests:

ttest_conf

TTest_conf returns the aggregated t-test confidence interval value for two independent samples iterated over the chart dimensions. This function applies to independent samples student's t-tests.

5 Functions in scripts and chart expressions

TTest_conf - chart function (grp, value [, sig[, eq_var]])

ttest_df

TTest_df() returns the aggregated student's t-test value (degrees of freedom) for two independent series of values iterated over the chart dimensions. This function applies to independent samples student's t-tests.

TTest_df - chart function (grp, value [, eq_var])

ttest_dif

TTest_dif() is a **numeric** function that returns the aggregated student's t-test mean difference for two independent series of values iterated over the chart dimensions. This function applies to independent samples student's t-tests.

TTest_dif - chart function (grp, value)

ttest_lower

TTest_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values iterated over the chart dimensions. This function applies to independent samples student's t-tests.

TTest_lower - chart function (grp, value [, sig[, eq_var]])

ttest_sig

TTest_sig() returns the aggregated student's t-test 2-tailed level of significance for two independent series of values iterated over the chart dimensions. This function applies to independent samples student's t-tests.

TTest_sig - chart function (grp, value [, eq_var])

ttest_sterr

TTest_sterr() returns the aggregated student's t-test standard error of the mean difference for two independent series of values iterated over the chart dimensions. This function applies to independent samples student's t-tests.

TTest_sterr - chart function (grp, value [, eq_var])

ttest_t

TTest_t() returns the aggregated t value for two independent series of values iterated over the chart dimensions. This function applies to independent samples student's t-tests.

TTest_t - chart function (grp, value [, eq_var])

ttest_upper

TTest_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values iterated over the chart dimensions. This function applies to independent samples student's t-tests.

TTest_upper - chart function (grp, value [, sig [, eq_var]])

5 Functions in scripts and chart expressions

Two independent weighted samples t-tests

The following functions to two independent samples student's t-tests where the input data series is given in weighted two-column format:

`ttestw_conf`

TTestw_conf() returns the aggregated t value for two independent series of values iterated over the chart dimensions. This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

```
TTestw_conf - chart function (weight, grp, value [, sig[, eq_var]])
```

`ttestw_df`

TTestw_df() returns the aggregated student's t-test df value (degrees of freedom) for two independent series of values iterated over the chart dimensions. This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

```
TTestw_df - chart function (weight, grp, value [, eq_var])
```

`ttestw_dif`

TTestw_dif() returns the aggregated student's t-test mean difference for two independent series of values iterated over the chart dimensions. This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

```
TTestw_dif - chart function ( weight, grp, value)
```

`ttestw_lower`

TTestw_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values iterated over the chart dimensions. This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

```
TTestw_lower - chart function (weight, grp, value [, sig[, eq_var]])
```

`ttestw_sig`

TTestw_sig() returns the aggregated student's t-test 2-tailed level of significance for two independent series of values iterated over the chart dimensions. This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

```
TTestw_sig - chart function ( weight, grp, value [, eq_var])
```

`ttestw_sterr`

TTestw_sterr() returns the aggregated student's t-test standard error of the mean difference for two independent series of values iterated over the chart dimensions. This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

```
TTestw_sterr - chart function (weight, grp, value [, eq_var])
```

`ttestw_t`

TTestw_t() returns the aggregated t value for two independent series of values iterated over the chart

5 Functions in scripts and chart expressions

dimensions.

```
TTestw_t - chart function (weight, grp, value [, eq_var])
```

ttestw_upper

TTestw_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values iterated over the chart dimensions. This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

```
TTestw_upper - chart function (weight, grp, value [, sig [, eq_var]])
```

One sample t-tests

The following functions apply to one-sample student's t-tests:

ttest1_conf

TTest1_conf() returns the aggregated confidence interval value for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests.

```
TTest1_conf - chart function (value [, sig])
```

ttest1_df

TTest1_df() returns the aggregated student's t-test df value (degrees of freedom) for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests.

```
TTest1_df - chart function (value)
```

ttest1_dif

TTest1_dif() returns the aggregated student's t-test mean difference for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests.

```
TTest1_dif - chart function (value)
```

ttest1_lower

TTest1_lower() returns the aggregated value for the lower end of the confidence interval for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests.

```
TTest1_lower - chart function (value [, sig])
```

ttest1_sig

TTest1_sig() returns the aggregated student's t-test 2-tailed level of significance for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests.

```
TTest1_sig - chart function (value)
```

ttest1_sterr

TTest1_sterr() returns the aggregated student's t-test standard error of the mean difference for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests.

```
TTest1_sterr - chart function (value)
```

ttest1_t

5 Functions in scripts and chart expressions

TTest1_t() returns the aggregated t value for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests.

```
TTest1_t - chart function (value)
```

ttest1_upper

TTest1_upper() returns the aggregated value for the upper end of the confidence interval for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests.

```
TTest1_upper - chart function (value [, sig])
```

One weighted sample t-tests

The following functions apply to one-sample student's t-tests where the input data series is given in weighted two-column format:

ttest1w_conf

TTest1w_conf() is a **numeric** function that returns the aggregated confidence interval value for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

```
TTest1w_conf - chart function (weight, value [, sig])
```

ttest1w_df

TTest1w_df() returns the aggregated student's t-test df value (degrees of freedom) for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

```
TTest1w_df - chart function (weight, value)
```

ttest1w_dif

TTest1w_dif() returns the aggregated student's t-test mean difference for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

```
TTest1w_dif - chart function (weight, value)
```

ttest1w_lower

TTest1w_lower() returns the aggregated value for the lower end of the confidence interval for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

```
TTest1w_lower - chart function (weight, value [, sig])
```

ttest1w_sig

TTest1w_sig() returns the aggregated student's t-test 2-tailed level of significance for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

```
TTest1w_sig - chart function (weight, value)
```

5 Functions in scripts and chart expressions

ttest1w_sterr

TTest1w_sterr() returns the aggregated student's t-test standard error of the mean difference for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

```
TTest1w_sterr - chart function (weight, value)
```

ttest1w_t

TTest1_t() returns the aggregated t value for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

```
TTest1w_t - chart function ( weight, value)
```

ttest1w_upper

TTest1w_upper() returns the aggregated value for the upper end of the confidence interval for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

```
TTest1w_upper - chart function (weight, value [, sig])
```

TTest_conf

This script function returns the aggregated t value for two independent series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest_conf (group, value [, sig = 0.025 [, eq_var = true]])
```

Arguments:

Argument	Description
group	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

5 Functions in scripts and chart expressions

Example:

```
LOAD Year, TTest_conf(Group, Value) as X from abc.csv group by Year;
```

TTest_conf - chart function

TTest_conf returns the aggregated t-test confidence interval value for two independent samples iterated over the chart dimensions.

This function applies to independent samples student's t-tests.

Syntax:

```
TTest_conf ( grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest_conf( Group, value )  
TTest_conf( Group, value, sig, false )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

5 Functions in scripts and chart expressions

TTest_df

This script function returns the aggregated student's t-test df value (degrees of freedom) for two independent series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest_df (group, value [, eq_var = true])
```

Arguments:

Argument	Description
group	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest_df(Group, value) as X from abc.csv group by Year;
```

TTest_df - chart function

TTest_df() returns the aggregated student's t-test value (degrees of freedom) for two independent series of values iterated over the chart dimensions.

This function applies to independent samples student's t-tests.

Syntax:

```
TTest_df (grp, value [, eq_var])
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as

5 Functions in scripts and chart expressions

Argument	Description
	specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest_df( Group, Value )  
TTest_df( Group, Value, false )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTest_dif

This script function returns the aggregated student's t-test mean difference for two independent series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest_dif (group, value [, eq_var = true])
```

Arguments:

Argument	Description
group	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

5 Functions in scripts and chart expressions

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest_dif(Group, Value) as X from abc.csv group by Year;
```

TTest_dif - chart function

TTest_dif() is a **numeric** function that returns the aggregated student's t-test mean difference for two independent series of values iterated over the chart dimensions.

This function applies to independent samples student's t-tests.

Syntax:

```
TTest_dif (grp, value [, eq_var] )
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest_dif( Group, value )  
TTest_dif( Group, value, false )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

5 Functions in scripts and chart expressions

TTest_lower

This script function returns the aggregated value for the lower end of the confidence interval for two independent series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest_lower (group, value [, sig = 0.025 [, eq_var = true]])
```

Arguments:

Argument	Description
group	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest_lower(Group, Value) as X from abc.csv group by Year;
```

TTest_lower - chart function

TTest_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values iterated over the chart dimensions.

This function applies to independent samples student's t-tests.

Syntax:

```
TTest_lower (grp, value [, sig [, eq_var]])
```

Arguments:

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as

5 Functions in scripts and chart expressions

Argument	Description
	specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest_lower( Group, Value )  
TTest_lower( Group, Value, Sig, false )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTest_sig

This script function returns the aggregated student's t-test 2-tailed level of significance for two independent series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest_sig (group, value [, eq_var = true])
```

Arguments:

Argument	Description
group	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .

5 Functions in scripts and chart expressions

Argument	Description
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, ttest_sig(Group, Value) as X from abc.csv group by Year;
```

TTest_sig - chart function

TTest_sig() returns the aggregated student's t-test 2-tailed level of significance for two independent series of values iterated over the chart dimensions.

This function applies to independent samples student's t-tests.

Syntax:

```
TTest_sig (grp, value [, eq_var])
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest_sig( Group, value )  
TTest_sig( Group, value, false )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTest_sterr

This script function returns the aggregated student's t-test standard error of the mean difference for two independent series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest_sterr (group, value [, eq_var = true])
```

Arguments:

Argument	Description
group	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest_sterr(Group, Value) as X from abc.csv group by Year;
```

TTest_sterr - chart function

TTest_sterr() returns the aggregated student's t-test standard error of the mean difference for two independent series of values iterated over the chart dimensions.

This function applies to independent samples student's t-tests.

Syntax:

```
TTest_sterr (grp, value [, eq_var])
```

Return data type: numeric

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest_sterr( Group, Value )  
TTest_sterr( Group, Value, false )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTest_t

This script function returns the aggregated t value for two independent series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest_t (group, value [, eq_var = true])
```

Arguments:

Argument	Description
group	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

5 Functions in scripts and chart expressions

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest_t(Group, value) as X from abc.csv group by Year;
```

TTest_t - chart function

TTest_t() returns the aggregated t value for two independent series of values iterated over the chart dimensions.

This function applies to independent samples student's t-tests.



*To be able to use this function you must load sample values in script using **crostable**.*

Syntax:

```
TTest_t(grp, value[, eq_var])
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

- ▢ *Creating a typical t-test report (page 312)*

5 Functions in scripts and chart expressions

TTest_upper

This script function returns the aggregated value for the upper end of the confidence interval for two independent series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest_upper (group, value [, sig = 0.025 [, eq_var = true]])
```

Arguments:

Argument	Description
group	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest_upper(Group, Value) as X from abc.csv group by Year;
```

TTest_upper - chart function

TTest_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values iterated over the chart dimensions.

This function applies to independent samples student's t-tests.

Syntax:

```
TTest_upper (grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest_upper( Group, Value )  
TTest_upper( Group, Value, sig, false )
```

See also:

▫ [Creating a typical t-test report \(page 312\)](#)

TTestw_conf

This script function returns the aggregated t value for two independent series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTestw_conf (weight, group, value [, sig = 0.025 [, eq_var = true]])
```

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
group	The field containing the names of each of the two sample groups. If a field name for the

5 Functions in scripts and chart expressions

Argument	Description
	group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTestw_conf(Weight, Group, Value) as X from abc.csv group by Year;
```

TTestw_conf - chart function

TTestw_conf() returns the aggregated t value for two independent series of values iterated over the chart dimensions.

This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

Syntax:

```
TTestw_conf (weight, grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025,

5 Functions in scripts and chart expressions

Argument	Description
	resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTestw_conf( weight, Group, value )  
TTestw_conf( weight, Group, value, sig, false )
```

See also:

▫ [Creating a typical t-test report \(page 312\)](#)

TTestw_df

This script function returns the aggregated student's t-test df value (degrees of freedom) for two independent series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTestw_df (weight, group, value [, eq_var = true])
```

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
group	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

5 Functions in scripts and chart expressions

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTestw_df(weight, Group, value) as X from abc.csv group by Year;
```

TTestw_df - chart function

TTestw_df() returns the aggregated student's t-test df value (degrees of freedom) for two independent series of values iterated over the chart dimensions.

This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

Syntax:

```
TTestw_df (weight, grp, value [, eq_var])
```

Return data type: numeric

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTestw_df( weight, Group, value )  
TTestw_df( weight, Group, value, false )
```

See also:

▫ [Creating a typical t-test report \(page 312\)](#)

5 Functions in scripts and chart expressions

TTestw_dif

This script function returns the aggregated student's t-test mean difference for two independent series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTestw_dif (weight, group, value [, eq_var = true])
```

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
group	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTestw_dif(weight, Group, Value) as X from abc.csv group by Year;
```

TTestw_dif - chart function

TTestw_dif() returns the aggregated student's t-test mean difference for two independent series of values iterated over the chart dimensions.

This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

Syntax:

```
TTestw_dif (weight, group, value)
```

Return data type: numeric

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTestw_dif( weight, Group, Value )  
TTestw_dif( weight, Group, value, false )
```

See also:

- *Creating a typical t-test report (page 312)*

TTestw_lower

This script function returns the aggregated value for the lower end of the confidence interval for two independent series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTestw_lower (weight, group, value [, sig = 0.025 [, eq_var = true]])
```

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
group	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as

5 Functions in scripts and chart expressions

Argument	Description
	specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTestw_lower(weight, Group, value) as X from abc.csv group by Year;
```

TTestw_lower - chart function

TTestw_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values iterated over the chart dimensions.

This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

Syntax:

```
TTestw_lower (weight, grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

5 Functions in scripts and chart expressions

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTestw_lower( weight, Group, Value )  
TTestw_lower( weight, Group, Value, sig, false )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTestw_sig

This script function returns the aggregated student's t-test 2-tailed level of significance for two independent series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTestw_sig (weight, group, value [, eq_var = true])
```

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
group	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTestw_sig(weight, Group, Value) as X from abc.csv group by Year;
```

TTestw_sig - chart function

TTestw_sig() returns the aggregated student's t-test 2-tailed level of significance for two independent series of values iterated over the chart dimensions.

This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

Syntax:

```
TTestw_sig ( weight, grp, value [, eq_var] )
```

Return data type: numeric

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTestw_sig( weight, Group, value )  
TTestw_sig( weight, Group, value, false )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTestw_sterr

This script function returns the aggregated student's t-test standard error of the mean difference for two independent series of values iterated over a number of records as defined by a **group by** clause.

5 Functions in scripts and chart expressions

Syntax:

```
TTestw_sterr (weight, group, value [, eq_var = true])
```

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
group	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTestw_sterr(weight, Group, value) as X from abc.csv group by Year;
```

TTestw_sterr - chart function

TTestw_sterr() returns the aggregated student's t-test standard error of the mean difference for two independent series of values iterated over the chart dimensions.

This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

Syntax:

```
TTestw_sterr (weight, grp, value [, eq_var])
```

Return data type: numeric

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .

5 Functions in scripts and chart expressions

Argument	Description
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTestw_sterr( weight, Group, Value )  
TTestw_sterr( weight, Group, Value, false )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTestw_t

This script function returns the aggregated t value for two independent series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTestw_t (weight, group, value [, eq_var = true])
```

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
group	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .

5 Functions in scripts and chart expressions

Argument	Description
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LLoad Year, TTestw_t(weight, Group, value) as X from abc.csv group by Year;
```

TTestw_t - chart function

TTestw_t() returns the aggregated t value for two independent series of values iterated over the chart dimensions.

This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

Syntax:

```
ttestw_t (weight, grp, value [, eq_var])
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

5 Functions in scripts and chart expressions

TTestw_t(weight, Group, Value)
TTestw_t(weight, Group, Value, false)

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTestw_upper

This script function returns the aggregated value for the upper end of the confidence interval for two independent series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTestw_upper (weight, group, value [, sig = 0.025 [, eq_var = true]])
```

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
group	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTestw_upper(weight, Group, Value) as X from abc.csv group by Year;
```

TTestw_upper - chart function

TTestw_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values iterated over the chart dimensions.

5 Functions in scripts and chart expressions

This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

Syntax:

```
TTestw_upper (weight, grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTestw_upper( weight, Group, value )  
TTestw_upper( weight, Group, value, sig, false )
```

See also:

▫ [Creating a typical t-test report \(page 312\)](#)

TTest1_conf

This script function returns the aggregated t value for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest1_conf (value [, sig = 0.025 ])
```

5 Functions in scripts and chart expressions

Arguments:

Argument	Description
value	The values should be returned by value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest1_conf(Value) as X from abc.csv group by Year;
```

TTest1_conf - chart function

TTest1_conf() returns the aggregated confidence interval value for a series of values iterated over the chart dimensions.

This function applies to one-sample student's t-tests.

Syntax:

```
TTest1_conf (value [, sig ])
```

Return data type: numeric

Arguments:

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest1_conf( value )  
TTest1_conf( value, 0.005 )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTest1_df

This script function returns the aggregated student's t-test df value (degrees of freedom) for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest1_df (value)
```

Arguments:

Argument	Description
value	The values should be returned by value .

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest1_df(Value) as X from abc.csv group by Year;
```

TTest1_df - chart function

TTest1_df() returns the aggregated student's t-test df value (degrees of freedom) for a series of values iterated over the chart dimensions.

This function applies to one-sample student's t-tests.

Syntax:

```
TTest1_df (value)
```

Return data type: numeric

Arguments:

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .

Limitations:

5 Functions in scripts and chart expressions

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1_df( value )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTest1_dif

This script function returns the aggregated student's t-test mean difference for a series of values iterated over a number of records as defined by **agroup by** clause.

Syntax:

```
TTest1_dif (value)
```

Arguments:

Argument	Description
value	The values should be returned by value .

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest1_dif(value) as X from abc.csv group by Year;
```

TTest1_dif - chart function

TTest1_dif() returns the aggregated student's t-test mean difference for a series of values iterated over the chart dimensions.

This function applies to one-sample student's t-tests.

Syntax:

```
TTest1_dif (value)
```

Return data type: numeric

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1_dif( value )
```

See also:

- *Creating a typical t-test report (page 312)*

TTest1_lower

This script function returns the aggregated value for the lower end of the confidence interval for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest1_lower (value [, sig = 0.025 ])
```

Arguments:

Argument	Description
value	The values should be returned by value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest1_lower(value) as X from abc.csv group by Year;
```

TTest1_lower - chart function

TTest1_lower() returns the aggregated value for the lower end of the confidence interval for a series of values iterated over the chart dimensions.

5 Functions in scripts and chart expressions

This function applies to one-sample student's t-tests.

Syntax:

```
TTest1_lower (value [, sig])
```

Arguments:

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest1_lower( value )  
TTest1_lower( value, 0.005 )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTest1_sig

This script function returns the aggregated student's t-test 2-tailed level of significance for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest1_sig (value)
```

Arguments:

Argument	Description
value	The values should be returned by value .

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest1_sig(Value) as X from abc.csv group by Year;
```

TTest1_sig - chart function

TTest1_sig() returns the aggregated student's t-test 2-tailed level of significance for a series of values iterated over the chart dimensions.

This function applies to one-sample student's t-tests.

Syntax:

```
TTest1_sig (value)
```

Return data type: numeric

Arguments:

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1_sig( value )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTest1_sterr

This script function returns the aggregated student's t-test standard error of the mean difference for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest1_sterr (value)
```

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
value	The values should be returned by value .

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest1_sterr(Value) as X from abc.csv group by Year;
```

TTest1_sterr - chart function

TTest1_sterr() returns the aggregated student's t-test standard error of the mean difference for a series of values iterated over the chart dimensions.

This function applies to one-sample student's t-tests.

Syntax:

```
TTest1_sterr (value)
```

Return data type: numeric

Arguments:

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1_sterr( value )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTest1_t

This script function returns the aggregated t value for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest1_t (value)
```

Arguments:

Argument	Description
value	The values should be returned by value .

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, ttest1_t(Value) as X from abc.csv group by Year;
```

TTest1_t - chart function

TTest1_t() returns the aggregated t value for a series of values iterated over the chart dimensions.

This function applies to one-sample student's t-tests.

Syntax:

```
TTest1_t (value)
```

Return data type: numeric

Arguments:

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1_t( value )
```

TTest1_upper

This script function returns the aggregated value for the upper end of the confidence interval for a series of values iterated over a number of records as defined by a **group by** clause.

5 Functions in scripts and chart expressions

Syntax:

```
TTest1_upper (value [, sig = 0.025 ])
```

Arguments:

Argument	Description
value	The values should be returned by value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest1_upper(Value) as X from abc.csv group by Year;
```

TTest1_upper - chart function

TTest1_upper() returns the aggregated value for the upper end of the confidence interval for a series of values iterated over the chart dimensions.

This function applies to one-sample student's t-tests.

Syntax:

```
TTest1_upper (value [, sig])
```

Return data type: numeric

Arguments:

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest1_upper( value )  
TTest1_upper( value, 0.005 )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTest1w_conf

This script function returns the aggregated t value for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest1w_conf (weight, value [, sig = 0.025 ])
```

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
value	The values should be returned by value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest1w_conf(weight, value) as X from abc.csv group by Year;
```

TTest1w_conf - chart function

TTest1w_conf() is a **numeric** function that returns the aggregated confidence interval value for a series of values iterated over the chart dimensions.

This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

Syntax:

```
TTest1w_conf (weight, value [, sig ])
```

5 Functions in scripts and chart expressions

Return data type: numeric

Arguments:

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest1w_conf( weight, value )  
TTest1w_conf( weight, value, 0.005 )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTest1w_df

This script function returns the aggregated student's t-test df value (degrees of freedom) for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest1w_df (weight, value)
```

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
value	The values should be returned by value .

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest1w_df(Weight, Value) as X from abc.csv group by Year;
```

TTest1w_df - chart function

TTest1w_df() returns the aggregated student's t-test df value (degrees of freedom) for a series of values iterated over the chart dimensions.

This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

Syntax:

```
TTest1w_df (weight, value)
```

Return data type: numeric

Arguments:

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1w_df( weight, Value )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTest1w_dif

This script function returns the aggregated student's t-test mean difference for a series of values iterated over a number of records as defined by **agroup by** clause.

5 Functions in scripts and chart expressions

Syntax:

```
TTest1w_dif (weight, value)
```

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
value	The values should be returned by value .

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest1w_dif(weight, value) as X from abc.csv group by Year;
```

TTest1w_dif - chart function

TTest1w_dif() returns the aggregated student's t-test mean difference for a series of values iterated over the chart dimensions.

This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

Syntax:

```
TTest1w_dif (weight, value)
```

Return data type: numeric

Arguments:

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .

Limitations:

5 Functions in scripts and chart expressions

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1w_dif( weight, value )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTest1w_lower

This script function returns the aggregated value for the lower end of the confidence interval for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest1w_lower (weight, value [, sig = 0.025 ])
```

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
value	The values should be returned by value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest1w_lower(weight, value) as X from abc.csv group by Year;
```

TTest1w_lower - chart function

TTest1w_lower() returns the aggregated value for the lower end of the confidence interval for a series of values iterated over the chart dimensions.

This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

Syntax:

5 Functions in scripts and chart expressions

```
TTest1w_lower (weight, value [, sig ])
```

Return data type: numeric

Arguments:

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest1w_lower( weight, value )
```

```
TTest1w_lower( weight, value, 0.005 )
```

See also:

▫ [Creating a typical t-test report \(page 312\)](#)

TTest1w_sig

This script function returns the aggregated student's t-test 2-tailed level of significance for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest1w_sig (weight, value)
```

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
value	The values should be returned by value .

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest1w_sig(weight, value) as X from abc.csv group by Year;
```

TTest1w_sig - chart function

TTest1w_sig() returns the aggregated student's t-test 2-tailed level of significance for a series of values iterated over the chart dimensions.

This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

Syntax:

```
TTest1w_sig (weight, value)
```

Return data type: numeric

Arguments:

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1w_sig( weight, value )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTest1w_sterr

This script function returns the aggregated student's t-test standard error of the mean difference for two independent series of values iterated over a number of records as defined by a **group by** clause.

5 Functions in scripts and chart expressions

Syntax:

```
TTestw_sterr (weight, group, value [, eq_var = true])
```

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
group	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTestw_sterr(weight, Group, value) as X from abc.csv group by Year;
```

TTest1w_sterr - chart function

TTest1w_sterr() returns the aggregated student's t-test standard error of the mean difference for a series of values iterated over the chart dimensions.

This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

Syntax:

```
TTest1w_sterr (weight, value)
```

Return data type: numeric

Arguments:

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .

5 Functions in scripts and chart expressions

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1w_sterr( weight, value )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTest1w_t

This script function returns the aggregated t value for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest1w_t (weight, value)
```

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
value	The values should be returned by value .

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest1w_t(weight, value) as X from abc.csv group by Year;
```

TTest1w_t - chart function

TTest1w_t() is a **numeric** function that returns the aggregated t value for a series of values iterated over the chart dimensions. This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

Syntax:

```
TTest1w_t ( weight, value)
```

Return data type: numeric

Arguments:

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1w_t( weight, value )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

TTest1w_upper

This script function returns the aggregated value for the upper end of the confidence interval for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
TTest1w_upper (weight, value [, sig = 0.025 ])
```

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
value	The values should be returned by value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, TTest1w_upper(weight, Value) as X from abc.csv group by Year;
```

TTest1w_upper - chart function

TTest1w_upper() returns the aggregated value for the upper end of the confidence interval for a series of values iterated over the chart dimensions.

This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

Syntax:

```
TTest1w_upper (weight, value [, sig])
```

Return data type: numeric

Arguments:

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:


```
TTest1w_upper( weight, value )  
TTest1w_upper( weight, value, 0.005 )
```

See also:

- [Creating a typical t-test report \(page 312\)](#)

Z-test functions in charts

A statistical examination of two population means. A two sample z-test examines whether two samples are different and is commonly used when two normal distributions have **known variances** and when an experiment uses a **large sample size**.

In the following the z-test statistical test functions are grouped according the type of input data series that applies to the function.

Z-test functions in the data load script

One column format functions

The following five functions apply to z-tests.

ZTest_z

This script function returns the aggregated z value for a series of values iterated over a number of records as defined by a **group by** clause.

```
ZTest_z (value [, sigma])
```

ZTest_sig

This script function returns the aggregated z-test 2-tailed level of significance for a series of values iterated over a number of records as defined by a **group by** clause.

```
ZTest_sig (value [, sigma])
```

ZTest_dif

This script function returns the aggregated z-test mean difference for a series of values iterated over a number of records as defined by a **group by** clause.

```
ZTest_dif (value [, sigma])
```

ZTest_sterr

This script function returns the aggregated z-test standard error of the mean difference for a series of values iterated over a number of records as defined by a **group by** clause.

```
ZTest_sterr (value [, sigma])
```

ZTest_conf

This script function returns the aggregated z value for a series of values iterated over a number of records as defined by a **group by** clause.

5 Functions in scripts and chart expressions

```
ZTest_conf (value [, sigma [, sig = 0.025 ]])
```

Weighted two-column format functions

The following five functions apply to z-tests where the input data series is given in weighted two-column format.

ZTestw_z

This script function returns the aggregated z value for a series of values iterated over a number of records as defined by a **group by** clause.

```
ZTestw_z (weight, value [, sigma])
```

ZTestw_sig

This script function returns the aggregated z-test 2-tailed level of significance for a series of values iterated over a number of records as defined by a **group by** clause.

```
ZTestw_sig (weight, value [, sigma])
```

ZTestw_dif

This script function returns the aggregated z-test mean difference for a series of values iterated over a number of records as defined by a **group by** clause.

```
ZTestw_dif (weight, value [, sigma])
```

ZTestw_sterr

This script function returns the aggregated z-test standard error of the mean difference for a series of values iterated over a number of records as defined by a **group by** clause.

```
ZTestw_sterr (weight, value [, sigma])
```

ZTestw_conf

This script function returns the aggregated z value for a series of values iterated over a number of records as defined by a **group by** clause.

```
ZTestw_conf (weight, value [, sigma [, sig = 0.025 ]])
```

Z-test functions in chart expressions

One column format functions

The following functions apply to z-tests with simple input data series:

ztest_conf

ZTest_conf() returns the aggregated z value for a series of values iterated over the chart dimensions.

```
ZTest_conf - chart function (value [, sigma [, sig ]])
```

ztest_dif

ZTest_dif() returns the aggregated z-test mean difference for a series of values iterated over the chart dimensions.

5 Functions in scripts and chart expressions

ZTest_dif - chart function (value [, sigma])

ztest_sig

ZTest_sig() returns the aggregated z-test 2-tailed level of significance for a series of values iterated over the chart dimensions.

ZTest_sig - chart function (value [, sigma])

ztest_sterr

ZTest_sterr() returns the aggregated z-test standard error of the mean difference for a series of values iterated over the chart dimensions.

ZTest_sterr - chart function (value [, sigma])

ztest_z

ZTest_z() returns the aggregated z value for a series of values iterated over the chart dimensions.

ZTest_z - chart function (value [, sigma])

ztest_lower

ZTest_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values iterated over the chart dimensions.

ZTest_lower - chart function (grp, value [, sig [, eq_var]])

ztest_upper

ZTest_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values iterated over the chart dimensions.

ZTest_upper - chart function (grp, value [, sig [, eq_var]])

Weighted two-column format functions

The following functions apply to z-tests where the input data series is given in weighted two-column format.

ztestw_conf

ZTestw_conf() returns the aggregated z confidence interval value for a series of values iterated over the chart dimensions.

ZTestw_conf - chart function (weight, value [, sigma [, sig]])

ztestw_dif

ZTestw_dif() returns the aggregated z-test mean difference for a series of values iterated over the chart dimensions.

ZTestw_dif - chart function (weight, value [, sigma])

ztestw_lower

ZTestw_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values iterated over the chart dimensions.

5 Functions in scripts and chart expressions

ZTestw_lower - chart function (weight, value [, sigma])

ztestw_sig

ZTestw_sig() returns the aggregated z-test 2-tailed level of significance for a series of values iterated over the chart dimensions.

ZTestw_sig - chart function (weight, value [, sigma])

ztestw_sterr

ZTestw_sterr() returns the aggregated z-test standard error of the mean difference for a series of values iterated over the chart dimensions.

ZTestw_sterr - chart function (weight, value [, sigma])

ztestw_upper

ZTestw_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values iterated over the chart dimensions.

ZTestw_upper - chart function (weight, value [, sigma])

ztestw_z

ZTestw_z() returns the aggregated z value for a series of values iterated over the chart dimensions.

ZTestw_z - chart function (weight, value [, sigma])

ZTest_z

This script function returns the aggregated z value for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

ZTest_z (value [, sigma])

A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.

Arguments:

Argument	Description
value	The values should be returned by value .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, ZTest_z(Value-TestValue) as X from abc.csv group by Year;
```

ZTest_z - chart function

ZTest_z() returns the aggregated z value for a series of values iterated over the chart dimensions.

Syntax:

```
ZTest_z(value[, sigma])
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that mean from the sample values.
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTest_z( Value-TestValue )
```

See also:

- ▢ [Examples of how to use z-test functions \(page 315\)](#)

ZTest_sig

This script function returns the aggregated z-test 2-tailed level of significance for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
ZTest_sig (value [, sigma])
```

A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.

5 Functions in scripts and chart expressions

Arguments:

Argument	Description
value	The values should be returned by value .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, ZTest_sig(Value-TestValue) as x from abc.csv group by Year;
```

ZTest_sig - chart function

ZTest_sig() returns the aggregated z-test 2-tailed level of significance for a series of values iterated over the chart dimensions.

Syntax:

```
ZTest_sig(value[, sigma])
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that mean from the sample values.
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTest_sig(Value-TestValue)
```

See also:

- ▢ [Examples of how to use z-test functions \(page 315\)](#)

ZTest_dif

This script function returns the aggregated z-test mean difference for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
ZTest_dif (value [, sigma])
```

A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.

Arguments:

Argument	Description
value	The values should be returned by value .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, ZTest_dif(Value-TestValue) as X from abc.csv group by Year
```

ZTest_dif - chart function

ZTest_dif() returns the aggregated z-test mean difference for a series of values iterated over the chart dimensions.

Syntax:

```
ZTest_dif (value[, sigma])
```

Return data type: numeric

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
value	The sample values to be evaluated. A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that mean from the sample values.
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTest_dif(Value-TestValue)
```

See also:

- *Examples of how to use z-test functions (page 315)*

ZTest_sterr

This script function returns the aggregated z-test standard error of the mean difference for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
ZTest_sterr (value [, sigma])
```

A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.

Arguments:

Argument	Description
value	The values should be returned by value .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

5 Functions in scripts and chart expressions

```
LOAD Year, ZTest_sterr(Value-TestValue) as X from abc.csv group by Year;
```

ZTest_sterr - chart function

ZTest_sterr() returns the aggregated z-test standard error of the mean difference for a series of values iterated over the chart dimensions.

Syntax:

```
ZTest_sterr (value [, sigma])
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that mean from the sample values.
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTest_sterr(Value-TestValue)
```

See also:

- ▢ [Examples of how to use z-test functions \(page 315\)](#)

ZTest_conf

This script function returns the aggregated z value for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
ZTest_conf (value [, sigma [, sig = 0.025 ]])
```

A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
value	The values should be returned by value .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, ZTest_conf(Value-TestValue) as X from abc.csv group by Year;
```

ZTest_conf - chart function

ZTest_conf() returns the aggregated z value for a series of values iterated over the chart dimensions.

Syntax:

```
ZTest_conf(value[, sigma[, sig]])
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that mean from the sample values.
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTest_conf(Value-TestValue)
```

See also:

- [Examples of how to use z-test functions \(page 315\)](#)

ZTest_lower - chart function

ZTest_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values iterated over the chart dimensions.

Syntax:

```
ZTest_lower (grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
ZTest_lower( Group, value )  
ZTest_lower( Group, value, sig, false )
```

See also:

- [Examples of how to use z-test functions \(page 315\)](#)

5 Functions in scripts and chart expressions

ZTest_upper - chart function

ZTest_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values iterated over the chart dimensions.

This function applies to independent samples student's t-tests.

Syntax:

```
ZTest_upper (grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
ZTest_upper( Group, value )  
ZTest_upper( Group, value, sig, false )
```

See also:

- ▢ [Examples of how to use z-test functions \(page 315\)](#)

ZTestw_z

This script function returns the aggregated z value for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

5 Functions in scripts and chart expressions

```
ZTestw_z (weight, value [, sigma])
```

A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
value	The values should be returned by value .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, ZTestw_z(Weight,Value-TestValue) as X from abc.csv group by Year;
```

ZTestw_z - chart function

ZTestw_z() returns the aggregated z value for a series of values iterated over the chart dimensions.

This function applies to z-tests where the input data series is given in weighted two-column format.

Syntax:

```
ZTestw_z (weight, value [, sigma])
```

Return data type: numeric

Arguments:

Argument	Description
value	The values should be returned by value . A sample mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.
weight	Each sample value in value can be counted one or more times according to a corresponding weight value in weight .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

5 Functions in scripts and chart expressions

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTestw_z( weight, value-TestValue)
```

See also:

- [Examples of how to use z-test functions \(page 315\)](#)

ZTestw_sig

This script function returns the aggregated z-test 2-tailed level of significance for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
ZTestw_sig (weight, value [, sigma])
```

A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
value	The values should be returned by value .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, ZTestw_sig(weight,value-TestValue) as X from abc.csv group by Year;
```

ZTestw_sig - chart function

ZTestw_sig() returns the aggregated z-test 2-tailed level of significance for a series of values iterated over the chart dimensions.

This function applies to z-tests where the input data series is given in weighted two-column format.

Syntax:

```
ZTestw_sig (weight, value [, sigma])
```

Return data type: numeric

Arguments:

Argument	Description
value	The values should be returned by value . A sample mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.
weight	Each sample value in value can be counted one or more times according to a corresponding weight value in weight .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTestw_sig( weight, value-Testvalue)
```

See also:

- [Examples of how to use z-test functions \(page 315\)](#)

ZTestw_dif

This script function returns the aggregated z-test mean difference for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
ZTestw_dif (weight, value [, sigma])
```

A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
value	The values should be returned by value .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, ZTestw_dif(weight,Value-TestValue) as X from abc.csv group by Year;
```

ZTestw_dif - chart function

ZTestw_dif() returns the aggregated z-test mean difference for a series of values iterated over the chart dimensions.

This function applies to z-tests where the input data series is given in weighted two-column format.

Syntax:

```
ZTestw_dif ( weight, value [, sigma]
```

Return data type: numeric

Arguments:

Argument	Description
value	The values should be returned by value . A sample mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.
weight	Each sample value in value can be counted one or more times according to a corresponding weight value in weight .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTestw_dif( weight, value-TestValue)
```


See also:

- [Examples of how to use z-test functions \(page 315\)](#)

ZTestw_sterr

This script function returns the aggregated z-test standard error of the mean difference for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
ZTestw_sterr (weight, value [, sigma])
```

A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
value	The values should be returned by value .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, ZTestw_sterr(weight,value-TestValue) as X from abc.csv group by Year;
```

ZTestw_sterr - chart function

ZTestw_sterr() returns the aggregated z-test standard error of the mean difference for a series of values iterated over the chart dimensions.

This function applies to z-tests where the input data series is given in weighted two-column format.

Syntax:

```
ZTestw_sterr (weight, value [, sigma])
```

Return data type: numeric

Arguments:

Argument	Description
value	The values should be returned by value . A sample mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.
weight	Each sample value in value can be counted one or more times according to a corresponding weight value in weight .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTestw_sterr( weight, value-Testvalue)
```

See also:

- [Examples of how to use z-test functions \(page 315\)](#)

ZTestw_conf

This script function returns the aggregated z value for a series of values iterated over a number of records as defined by a **group by** clause.

Syntax:

```
ZTestw_conf (weight, value [, sigma [, sig = 0.025 ]])
```

A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.

Arguments:

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
value	The values should be returned by value .

5 Functions in scripts and chart expressions

Argument	Description
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in **value** will result in the function returning NULL.

Example:

```
LOAD Year, ZTestw_conf(Weight,Value-Testvalue) as X from abc.csv group by Year;
```

ZTestw_conf - chart function

ZTestw_conf() returns the aggregated z confidence interval value for a series of values iterated over the chart dimensions.

This function applies to z-tests where the input data series is given in weighted two-column format.

Syntax:

```
ZTest_conf(weight, value[, sigma[, sig]])
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that mean from the sample values.
weight	Each sample value in value can be counted one or more times according to a corresponding weight value in weight .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

5 Functions in scripts and chart expressions

ZTestw_conf(weight, Value-TestValue)

See also:

- ▢ [Examples of how to use z-test functions \(page 315\)](#)

ZTestw_lower - chart function

ZTestw_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values iterated over the chart dimensions.

Syntax:

```
ZTestw_lower (grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
ZTestw_lower( Group, value )  
ZTestw_lower( Group, value, sig, false )
```

See also:

- ▢ [Examples of how to use z-test functions \(page 315\)](#)

5 Functions in scripts and chart expressions

ZTestw_upper - chart function

ZTestw_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values iterated over the chart dimensions.

This function applies to independent samples student's t-tests.

Syntax:

```
ZTestw_upper (grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
ZTestw_upper( Group, Value )  
ZTestw_upper( Group, Value, sig, false )
```

See also:

▢ [Examples of how to use z-test functions \(page 315\)](#)

Statistical test function examples for charts

This section includes examples of statistical test functions as applied to charts.

Examples of how to use chi2-test functions

The chi2-test functions are used to find values associated with chi squared statistical analysis. This section describes how to build visualizations using sample data to find the values of the chi-squared distribution test functions available in Qlik Sense. Please refer to the individual chi2-test chart function topics for descriptions of syntax and arguments.

Loading the data for the samples

There are three sets of sample data describing three different statistical samples to be loaded into the script.


Do the following:

1. Create a new app.
2. In the data load, enter the following:

```
// Sample_1 data is pre-aggregated... Note: make sure you set your DecimalSep='.' at the top
of the script.
Sample_1:
LOAD * inline [
Grp,Grade,Count
I,A,15
I,B,7
I,C,9
I,D,20
I,E,26
I,F,19
II,A,10
II,B,11
II,C,7
II,D,15
II,E,21
II,F,16
];
// Sample_2 data is pre-aggregated: If raw data is used, it must be aggregated using count
()...
Sample_2:
LOAD * inline [
Sex,Opinion,OpCount
1,2,58
1,1,11
1,0,10
2,2,35
2,1,25
2,0,23 ] (delimiter is ',');
// Sample_3a data is transformed using the crosstable statement...
Sample_3a:
crosstable(Gender, Actual) LOAD
Description,
[Men (Actual)] as Men,
[Women (Actual)] as Women;
LOAD * inline [
Men (Actual),Women (Actual),Description
58,35,Agree
11,25,Neutral
10,23,Disagree ] (delimiter is ',');
// Sample_3b data is transformed using the crosstable statement...
```

5 Functions in scripts and chart expressions



```
Sample_3b:
crosstable(Gender, Expected) LOAD
Description,
[Men (Expected)] as Men,
[Women (Expected)] as Women;
LOAD * inline [
Men (Expected),Women (Expected),Description
45.35,47.65,Agree
17.56,18.44,Neutral
16.09,16.91,Disagree ] (delimiter is ',');
// Sample_3a and Sample_3b will result in a (fairly harmless) synthetic key...
```

3. Click  to load data.

Creating the chi2-test chart function visualizations

Example: Sample 1

Do the following:

1. In the data load editor, click  to go to the app view and then click the sheet you created before. The sheet view is opened.
2. Click  **Edit** to edit the sheet.
3. From **Charts** add a table, and from **Fields** add Grp, Grade, and Count as dimensions. This table shows the sample data.
4. Add another table with the following expression as a dimension:
`valueList('p','df','chi2')`
This uses the synthetic dimensions function to create labels for the dimensions with the names of the three chi2-test functions.
5. Add the following expression to the table as a measure:
`IF(ValueList('p','df','Chi2')='p',Chi2Test_p(Grp,Grade,Count),
IF(ValueList('p','df','Chi2')='df',Chi2Test_df(Grp,Grade,Count),
Chi2Test_Chi2(Grp,Grade,Count)))`
This has the effect of putting the resulting value of each chi2-test function in the table next to its associated synthetic dimension.
6. Set the **Number formatting** of the measure to **Number** and **3Significant figures**.



In the expression for the measure, you could use the following expression instead: `Pick(Match(ValueList('p','df','Chi2'),'p','df','Chi2'),Chi2Test_p(Grp,Grade,Count),Chi2Test_df(Grp,Grade,Count),Chi2Test_Chi2(Grp,Grade,Count))`

Result

The resulting table for the chi2-test functions for the Sample 1 data will contain the following values:

p	df	Chi2
0.820	5	2.21

Example: Sample 2

5 Functions in scripts and chart expressions

Do the following:

1. In the sheet you were editing in the example Sample 1, from **Charts** add a table, and from **Fields** add Sex, Opinion, and OpCount as dimensions.
2. Make a copy of the results table from Sample 1 using the **Copy** and **Paste** commands. Edit the expression in the measure and replace the arguments in all three chi2-test functions with the names of the fields used in the Sample 2 data, for example: `chi2Test_p(Sex,Opinion,OpCount)`.

Result

The resulting table for the chi2-test functions for the Sample 2 data will contain the following values:

p	df	Chi2
0.000309	2	16.2

Example: Sample 3

Do the following:

1. Create two more tables in the same way as in the examples for Sample 1 and Sample 2 data. In the dimensions table, use the following fields as dimensions: Gender, Description, Actual, and Expected.
2. In the results table, use the names of the fields used in the Sample 3 data, for example: `chi2Test_p(Gender,Description,Actual,Expected)`.

Result

The resulting table for the chi2-test functions for the Sample 3 data will contain the following values:

p	df	Chi2
0.000308	2	16.2

Creating a typical t-test report

A typical student t-test report can include tables with **Group Statistics** and **Independent Samples Test** results. In the following sections we will build these tables using Qlik Sense test functions applied to two independent groups of samples, Observation and Comparison. The corresponding tables for these samples would look like this:

Group Statistics

Type	N	Mean	Standard Deviation	Standard Error Mean
Comparison	20	11.95	14.61245	3.2674431
Observation	20	27.15	12.507997	2.7968933

Independent Sample Test

5 Functions in scripts and chart expressions

	t	df	Sig. (2- tailed)	Mean Difference	Standard Error Difference	95% Confidence Interval of the Difference (Lower)	95% Confidence Interval of the Difference (Upper)
Equal Variance not Assumed	3.534	37.116717335823	0.001	15.2	4.30101	6.48625	23.9137
Equal Variance Assumed	3.534	38	0.001	15.2	4.30101	6.49306	23.9069


Loading the sample data

Do the following:

1. Create a new app with a new sheet and open that sheet.
2. Enter the following in the data load editor:



```
Table1:
crosstable LOAD recno() as ID, * inline [
observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

In this load script, **recno()** is included because **crosstable** requires three arguments. So, **recno()** simply provides an extra argument, in this case an ID for each row. Without it, **Comparison** sample values would not be loaded.

3. Click  to load data.

5 Functions in scripts and chart expressions

Creating the Group Statistics table

1. In the data load editor, click  to go to app view, and then click the sheet you created before. This opens the sheet view.
2. Click  **Edit** to edit the sheet.
3. From **Charts**, add a table, and from **Fields**, add the following expressions as measures:

Label	Expression
N	Count(Value)
Mean	Avg(Value)
Standard Deviation	Std(Value)
Standard Error Mean	Sterr(Value)


4. Add **Type** as a dimension to the table.
5. Click **Sorting** and move **Type** to the top of the sorting list.

Result

A **Group Statistics** table for these samples would look like this:

Type	N	Mean	Standard Deviation	Standard Error Mean
Comparison	20	11.95	14.61245	3.2674431
Observation	20	27.15	12.507997	2.7968933

Creating the Two Independent Sample Student's T-test table

1. Click  **Edit** to edit the sheet.
2. Add the following expression as a dimension to the table. =valueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1))
3. From **Charts** add a table with the following expressions as measures:

Label	Expression
conf	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_conf(Type, Value),TTest_conf(Type, Value, 0))
t	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_t(Type, Value),TTest_t(Type, Value, 0))
df	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_df(Type, Value),TTest_df(Type, Value, 0))
Sig. (2-tailed)	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_sig(Type, Value),TTest_sig(Type, Value, 0))
Mean Difference	TTest_dif(Type, Value)

5 Functions in scripts and chart expressions

Label	Expression
Standard Error Difference	<code>if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)), TTest_sterr(Type, Value), TTest_sterr(Type, Value, 0))</code>
95% Confidence Interval of the Difference (Lower)	<code>if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)), TTest_lower(Type, Value, (1-(95)/100)/2), TTest_lower(Type, Value, (1-(95)/100)/2, 0))</code>
95% Confidence Interval of the Difference (Upper)	<code>if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)), TTest_upper(Type, Value, (1-(95)/100)/2), TTest_upper (Type, Value, (1-(95)/100)/2, 0))</code>

Result

An **Independent Sample Test** table for these samples would look like this:

	t	df	Sig. (2-tailed)	Mean Difference	Standard Error Difference	95% Confidence Interval of the Difference (Lower)	95% Confidence Interval of the Difference (Upper)
Equal Variance not Assumed	3.53	37	0.001	15.2	4.30101	6.48625	23.9137
Equal Variance Assumed	3.53	38	0.001	15.2	4.30101	6.49306	23.9069

Examples of how to use z-test functions

The z-test functions are used to find values associated with z-test statistical analysis for large data samples, usually greater than 30, and where the variance is known. This section describes how to build visualizations using sample data to find the values of the z-test functions available in Qlik Sense. Please refer to the individual z-test chart function topics for descriptions of syntax and arguments.

Loading the sample data

The sample data used here is the same as that used in the t-test function examples. The sample data size would normally be considered too small for z-test analysis, but is sufficient for the purposes of illustrating the use of the different z-test functions in Qlik Sense.

Do the following:

5 Functions in scripts and chart expressions

1. Create a new app with a new sheet and open that sheet.




If you created an app for the t-test functions, you could use that and create a new sheet for these functions.

2. In the data load editor, enter the following:



```
Table1:
crosstable LOAD recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

In this load script, **recno()** is included because **crosstable** requires three arguments. So, **recno()** simply provides an extra argument, in this case an ID for each row. Without it, **Comparison** sample values would not be loaded.

3. Click  to load data.

Creating z-test chart function visualizations

Do the following:

1. In the data load editor, click  to go to app view, and then click the sheet you created when loading the data.
The sheet view is opened.
2. Click  **Edit** to edit the sheet.
3. From **Charts** add a table, and from **Fields** add Type as a dimension.
4. Add the following expressions to the table as measures.

Label	Expression
ZTest Conf	ZTest_conf(Value)
ZTest Dif	ZTest_dif(Value)

5 Functions in scripts and chart expressions

Label	Expression
ZTest Sig	ZTest_sig(Value)
ZTest Sterr	ZTest_sterr(Value)
ZTest Z	ZTest_z(Value)



You might wish to adjust the number formatting of the measures in order to see meaningful values. The table will be easier to read if you set number formatting on most of the measures to **Number>Simple**, instead of **Auto**. But for ZTest Sig, for example, use the number formatting: **Custom**, and then adjust the format pattern to **# ##**.

Result

The resulting table for the z-test functions for the sample data will contain the following values:

Type	ZTest Conf	ZTest Dif	ZTest Sig	ZTest Sterr	ZTest Z
Comparison	6.40	11.95	0.000123	3.27	3.66
Value	5.48	27.15	0.001	2.80	9.71

Creating z-testw chart function visualizations

The z-testw functions are for use when the input data series occurs in weighted two-column format. The expressions require a value for the argument weight. The examples here use the value 2 throughout, but you could use an expression, which would define a value for weight for each observation.

Examples and results:

Using the same sample data and number formatting as for the z-test functions, the resulting table for the z-testw functions will contain the following values:

Type	ZTestw Conf	ZTestw Dif	ZTestw Sig	ZTestw Sterr	ZTestw Z
Comparison	3.53	2.95	5.27e-005	1.80	3.88
Value	2.97	34.25	0	4.52	20.49

String aggregation functions

This section describes string-related aggregation functions.

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

String aggregation functions in the data load script

concat

This script function returns the aggregated string concatenation of all values of expression iterated over a number of records as defined by a **group by** clause.

```
concat ([ distinct ] expression [, delimiter [, sort-weight]])
```

FirstValue

This script function returns the first value in load order of expression over a number of records as defined by a **group by** clause.



This function is only available as a script function.

```
FirstValue (expression)
```

LastValue

This script function returns the last value in load order of expression over a number of records as defined by a **group by** clause.



This function is only available as a script function.

```
LastValue (expression)
```

MaxString

This script function returns the last text value of expression over a number of records, as defined by a **group by** clause.

```
MaxString (expression )
```

MinString

This script function returns the first text value of expression over a number of records, as defined by a **group by** clause.

```
MinString (expression )
```

String aggregation functions in charts

The following chart functions are available for aggregating strings in charts.

Concat

Concat() is used to combine string values. The function returns the aggregated string concatenation of all the values of the expression evaluated over each dimension.

```
Concat - chart function ({[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]]  
string[, delimiter[, sort_weight]])
```

MaxString

5 Functions in scripts and chart expressions

MaxString() finds string values in the expression or field and returns the last text value in the text sort order.

```
MaxString - chart function({[SetExpression] [TOTAL [<fld{, fld}>]]) expr)
```

MinString

MinString() finds string values in the expression or field and returns the first text value in the text sort order.

```
MinString - chart function({[SetExpression] [TOTAL [<fld {, fld}>]]) expr)
```

concat

This script function returns the aggregated string concatenation of all values of expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
concat ([ distinct ] expression [, delimiter [, sort-weight]])
```

Arguments:

Argument	Description
distinct	If the word distinct occurs before the expression, all duplicates are disregarded.
delimiter	Each value may be separated by the string found in delimiter.
sort-weight	The order of concatenation may be determined by sort-weight . sort_weight returns a numeric value where the lowest value will render the item to be sorted first.

Example:

```
LOAD Department, concat(Name,';') as NameList from abc.csv group by Department;
```

Concat - chart function

Concat() is used to combine string values. The function returns the aggregated string concatenation of all the values of the expression evaluated over each dimension.

Syntax:

```
Concat ({[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]] string[,  
delimiter[, sort_weight]])
```

Return data type: string

Arguments:

Argument	Description
string	The expression or field containing the string to be measured.
delimiter	Each value may be separated by the string found in the delimiter.

5 Functions in scripts and chart expressions

Argument	Description
sort-weight	The order of concatenation may be determined by sort-weight .
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Examples and results:

SalesGroup	Amount	Concat(Team)	Concat(TOTAL <SalesGroup> Team)
East	25000	Alpha	AlphaBetaDeltaGammaGamma
East	20000	BetaGammaGamma	AlphaBetaDeltaGammaGamma
East	14000	Delta	AlphaBetaDeltaGammaGamma
West	17000	Epsilon	EpsilonEtaThetaZeta
West	14000	Eta	EpsilonEtaThetaZeta
West	23000	Theta	EpsilonEtaThetaZeta
West	19000	Zeta	EpsilonEtaThetaZeta

Example	Result
Concat(Team)	The table is constructed from the dimensions SalesGroup and Amount, and variations on the measure Concat(Team). Ignoring the Totals result, note that even though there is data for eight values of Team spread across two values of SalesGroup, the only result of the measure Concat(Team) that concatenates more than one Team string value in the table is the row containing the dimension Amount 20000, which gives the result BetaGammaGamma. This is because there are three values for the Amount 20000 in the input data. All other results remain unconcatenated when the measure is spanned across the dimensions because there is only one value of Team for each combination of SalesGroup and Amount.
Concat ([DISTINCT, Team, ', '])	Beta, Gamma. because the DISTINCT qualifier means the duplicate Gamma result is disregarded. Also, the delimiter argument is defined as a comma followed

5 Functions in scripts and chart expressions

Example	Result
	by a space.
Concat (TOTAL <SalesGroup> Team)	All the string values for all values of Team are concatenated if the TOTAL qualifier is used. With the field selection <SalesGroup> specified, this divides the results into the two values of the dimension SalesGroup. For the SalesGroupEast, the results are AlphaBetaDeltaGammaGamma. For the SalesGroupWest, the results are EpsilonEtaThetaZeta.
Concat (TOTAL <SalesGroup> Team, ';', Amount)	By adding the argument for sort-weight : Amount, the results are ordered by the value of the dimension Amount. The results becomes DeltaBetaGammaGammaAlpha and EtaEpsilonZetaTheta.

Data used in example:

```
TeamData:
LOAD * inline [
SalesGroup|Team|Date|Amount
East|Gamma|01/05/2013|20000
East|Gamma|02/05/2013|20000
west|Zeta|01/06/2013|19000
East|Alpha|01/07/2013|25000
East|Delta|01/08/2013|14000
west|Epsilon|01/09/2013|17000
west|Eta|01/10/2013|14000
East|Beta|01/11/2013|20000
west|Theta|01/12/2013|23000
] (delimiter is '|');
```

FirstValue

This script function returns the first value in load order of expression over a number of records as defined by a **group by** clause.



This function is only available as a script function.

Syntax:

```
FirstValue ( expression)
```

Limitations:

If no text value is found, NULL is returned.

Example:

```
LOAD City, FirstValue(Name), as FirstName from abc.csv group by City;
```

LastValue

This script function returns the last value in load order of expression over a number of records as defined by a **group by** clause.



This function is only available as a script function.

Syntax:

```
LastValue ( expression )
```

Limitations:

If no text value is found, NULL is returned.

Example:

```
LOAD City, LastValue(Name), as FirstName from abc.csv group by City;
```

MaxString

This script function returns the last text value of expression over a number of records, as defined by a **group by** clause.

Syntax:

```
MaxString ( expression )
```

Limitations:

If no text value is found, NULL is returned.

Example:

```
LOAD Month, MaxString(Month) as LastSalesMonth from abc.csv group by Year;
```

MaxString - chart function

MaxString() finds string values in the expression or field and returns the last text value in the text sort order.

Syntax:

```
MaxString({[SetExpression] [TOTAL [<fld{, fld}>]]} expr)
```

Return data type: dual

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
TOTAL	<p>If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions.</p> <p>The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.</p>

Limitations:

If the expression contains no values with a string representation NULL is returned.

Examples and results:

SalesGroup	Amount	MaxString(Team)	MaxString(Date)
East	14000	Delta	2013/08/01
East	20000	Gamma	2013/11/01
East	25000	Alpha	2013/07/01
West	14000	Eta	2013/10/01
West	17000	Epsilon	2013/09/01
West	19000	Zeta	2013/06/01
West	23000	Theta	2013/12/01



This table represents all values of the dimension Customer with corresponding Product values. In an actual table visualization on a sheet, there will be a row for each value of Customer and Product.

Example	Result
MaxString (Team)	There are three values of 20000 for the dimension Amount: two of Gamma (on different dates), and one of Beta. The result of the measure MaxString (Team) is therefore Gamma, because this is the highest value in the sorted strings.
MaxString (Date))	2013/11/01 is the greatest Date value of the three associated with the dimension Amount. This assumes your script has the SET statement <code>SET DateFormat='YYYY-MM-DD';</code>

Data used in example:

```
TeamData:
LOAD * inline [
SalesGroup|Team|Date|Amount
East|Gamma|01/05/2013|20000
East|Gamma|02/05/2013|20000
west|Zeta|01/06/2013|19000
East|Alpha|01/07/2013|25000
East|Delta|01/08/2013|14000
west|Epsilon|01/09/2013|17000
west|Eta|01/10/2013|14000
East|Beta|01/11/2013|20000
west|Theta|01/12/2013|23000
] (delimiter is '|');
```

MinString

This script function returns the first text value of expression over a number of records, as defined by a **group by** clause.

Syntax:

```
MinString ( expression )
```

Limitations:

If no text value is found, NULL is returned.

Example:

```
LOAD Month, MinString(Month) as FirstSalesMonth from abc.csv group by Year;
```

MinString - chart function

MinString() finds string values in the expression or field and returns the first text value in the text sort order.

Syntax:

```
MinString ({ [SetExpression] [TOTAL [<fld {, fld}>]] } expr)
```

Return data type: dual

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.

5 Functions in scripts and chart expressions

Argument	Description
TOTAL	<p>If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions.</p> <p>The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.</p>

Examples and results:

SalesGroup	Amount	MinString(Team)	MinString(Date)
East	14000	Delta	2013/08/01
East	20000	Beta	2013/05/01
East	25000	Alpha	2013/07/01
West	14000	Eta	2013/10/01
West	17000	Epsilon	2013/09/01
West	19000	Zeta	2013/06/01
West	23000	Theta	2013/12/01

Examples	Results
MinString (Team)	There are three values of 20000 for the dimension Amount: two of Gamma (on different dates), and one of Beta. The result of the measure MinString (Team) is therefore Beta, because this is the first value in the sorted strings.
MinString (Date)	2013/11/01 is the earliest Date value of the three associated with the dimension Amount. This assumes your script has the SET statement <code>SET DateFormat='YYYY-MM-DD';</code>

Data used in example:

```
TeamData:
LOAD * inline [
SalesGroup|Team|Date|Amount
East|Gamma|01/05/2013|20000
East|Gamma|02/05/2013|20000
west|Zeta|01/06/2013|19000
East|Alpha|01/07/2013|25000
East|Delta|01/08/2013|14000
west|Epsilon|01/09/2013|17000
west|Eta|01/10/2013|14000
East|Beta|01/11/2013|20000
west|Theta|01/12/2013|23000
] (delimiter is '|');
```

Synthetic dimension functions

A synthetic dimension is created in the app from values generated from the synthetic dimension functions and not directly from fields in the data model. When values generated by a synthetic dimension function are used in a chart as a calculated dimension, this creates a synthetic dimension. Synthetic dimensions allow you to create, for example, charts with dimensions with values arising from your data, that is, dynamic dimensions.



Synthetic dimensions are not affected by selections.

The following synthetic dimension functions can be used in charts.

ValueList

ValueList() returns a set of listed values, which, when used in a calculated dimension, will form a synthetic dimension.

```
ValueList - chart function (v1 {, Expression})
```

ValueLoop

ValueLoop() returns a set of iterated values which, when used in a calculated dimension, will form a synthetic dimension.

```
ValueLoop - chart function(from [, to [, step ]])
```

ValueList - chart function

ValueList() returns a set of listed values, which, when used in a calculated dimension, will form a synthetic dimension.



*In charts with a synthetic dimension created with the **ValueList** function it is possible to reference the dimension value corresponding to a specific expression cell by restating the **ValueList** function with the same parameters in the chart expression. The function may of course be used anywhere in the layout, but apart from when used for synthetic dimensions it will only be meaningful inside an aggregation function.*



Synthetic dimensions are not affected by selections.

Syntax:

```
ValueList(v1 {, ...})
```

Return data type: dual

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
v1	Static value (usually a string, but can be a number).
{...}	Optional list of static values.

Examples and results:

Example	Result																																				
<code>ValueList('Number of Orders', 'Average Order Size', 'Total Amount')</code>	When used to create a dimension in a table, for example, this results in the three string values as row labels in the table. These can then be referenced in an expression.																																				
<code>=IF(ValueList ('Number of Orders', 'Average Order Size', 'Total Amount') = 'Number of Orders', count (SaleID), if(ValueList ('Number of Orders', 'Average Order Size', 'Total Amount') = 'Average Order Size', avg (Amount), sum (Amount)))</code>	<p>This expression takes the values from the created dimension and references them in a nested IF statement as input to three aggregation functions:</p> <table border="1"> <thead> <tr> <th colspan="4">ValueList()</th> </tr> <tr> <th>Created dimension</th> <th>Year</th> <th>Added expression</th> <th></th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> <td>522.00</td> </tr> <tr> <td>Number of Orders</td> <td>2012</td> <td></td> <td>5.00</td> </tr> <tr> <td>Number of Orders</td> <td>2013</td> <td></td> <td>7.00</td> </tr> <tr> <td>Average Order Size</td> <td>2012</td> <td></td> <td>13.20</td> </tr> <tr> <td>Average Order Size</td> <td>2013</td> <td></td> <td>15.43</td> </tr> <tr> <td>Total Amount</td> <td>2012</td> <td></td> <td>66.00</td> </tr> <tr> <td>Total Amount</td> <td>2013</td> <td></td> <td>108.00</td> </tr> </tbody> </table>	ValueList()				Created dimension	Year	Added expression					522.00	Number of Orders	2012		5.00	Number of Orders	2013		7.00	Average Order Size	2012		13.20	Average Order Size	2013		15.43	Total Amount	2012		66.00	Total Amount	2013		108.00
ValueList()																																					
Created dimension	Year	Added expression																																			
			522.00																																		
Number of Orders	2012		5.00																																		
Number of Orders	2013		7.00																																		
Average Order Size	2012		13.20																																		
Average Order Size	2013		15.43																																		
Total Amount	2012		66.00																																		
Total Amount	2013		108.00																																		

Data used in examples:

```

SalesPeople:
LOAD * INLINE [
SalesID|SalesPerson|Amount|Year
1|1|12|2013
2|1|23|2013
3|1|17|2013
4|2|9| 2013
5|2|14|2013
6|2|29|2013
7|2|4| 2013
8|1|15|2012
9|1|16|2012
10|2|11| 2012
11|2|17|2012
12|2|7| 2012
] (delimiter is '|');

```

ValueLoop - chart function

ValueLoop() returns a set of iterated values which, when used in a calculated dimension, will form a synthetic dimension.

The values generated will start with the **from** value and end with the **to** value including intermediate values in increments of step.



*In charts with a synthetic dimension created with the **ValueLoop** function it is possible to reference the dimension value corresponding to a specific expression cell by restating the **ValueLoop** function with the same parameters in the chart expression. The function may of course be used anywhere in the layout, but apart from when used for synthetic dimensions it will only be meaningful inside an aggregation function.*



Synthetic dimensions are not affected by selections.

Syntax:

```
ValueLoop(from [, to [, step ]])
```

Return data type: dual

Arguments:

Arguments	Description
from	Start value in the set of values to be generated.
to	End value in the set of values to be generated.
step	Size of increment between values.

Examples and results:

Example	Result
valueLoop (1, 10)	This creates a dimension in a table, for example, that can be used for purposes such as numbered labeling. The example here results in values numbered 1 to 10. These values can then be referenced in an expression.
valueLoop (2, 10,2)	This example results in values numbered 2, 4, 6, 8, and 10 because the argument step has a value of 2.

Nested aggregations

You may come across situations where you need to apply an aggregation to the result of another aggregation. This is referred to as nesting aggregations.

5 Functions in scripts and chart expressions

As a general rule, it is not allowed to nest aggregations in a Qlik Sense chart expression. Nesting is only allowed if you:

- Use the **TOTAL** qualifier in the inner aggregation function.



No more than 100 levels of nesting is allowed.

Nested aggregations with the TOTAL qualifier

Example:

You want to calculate the sum of the field **Sales**, but only include transactions with an **OrderDate** equal to the last year. The last year can be obtained via the aggregation function **Max (TOTAL Year (OrderDate))**.

The following aggregation would return the desired result:

```
Sum(If(Year(OrderDate)=Max(TOTAL Year(OrderDate)), Sales))
```

The inclusion of the **TOTAL** qualifier is absolutely necessary for this kind of nesting to be accepted by Qlik Sense, but then again also necessary for the desired comparison. This type of nesting need is quite common and is a good practice.

See also:

- [Aggr - chart function \(page 138\)](#)

5.2 Color functions

These functions can be used in expressions associated with setting and evaluating the color properties of chart objects, as well as in data load scripts.



*QlikView supports a number of color functions that are available in Qlik Sense for compatibility reasons, but use of them is not recommended: **blue, color, colormaphue, colormapjet, colormix1, colormix2, cyan, darkgray, green, lightblue, lightcyan, lightgray, lightmagenta, lightred, magenta, qliktechblue, qliktechgray, red, syscolor, white, yellow.***

ARGB

ARGB() is used in expressions to set or evaluate the color properties of a chart object, where the color is defined by a red component **r**, a green component **g**, and a blue component **b**, with an alpha factor (opacity) of **alpha**.

```
ARGB(alpha, r, g, b)
```

5 Functions in scripts and chart expressions

HSL

HSL() is used in expressions to set or evaluate the color properties of a chart object, where the color is defined by values of **hue**, **saturation**, and **luminosity** between 0 and 255.

```
HSL (hue, saturation, luminosity)
```

RGB

RGB() is used in expressions to set or evaluate the color properties of a chart object, where the color is defined by a red component **r**, a green component **g**, and a blue component **b**.

```
RGB (r, g, b)
```

ARGB

ARGB() is used in expressions to set or evaluate the color properties of a chart object, where the color is defined by a red component **r**, a green component **g**, and a blue component **b**, with an alpha factor (opacity) of **alpha**.

Syntax:

```
ARGB(alpha, r, g, b)
```

Return data type: dual

Arguments:

Argument	Description
alpha	Transparency value in the range 0 - 255. 0 corresponds to full transparency and 255 corresponds to full opacity.
r, g, b	Red, green, and blue component values. A color component of 0 corresponds to no contribution and one of 255 to full contribution.



All arguments must be expressions that resolve to integers in the range 0 to 255.

If interpreting the numeric component and formatting it in hexadecimal notation, the values of the color components are easier to see. For example, light green has the number 4 278 255 360, which in hexadecimal notation is FF00FF00. The first two positions 'FF' (255) denote the **alpha** factor. The next two positions '00' denote the amount of **red**, the next two positions 'FF' denote the amount of **green** and the final two positions '00' denote the amount of **blue**.

RGB

RGB() is used in expressions to set or evaluate the color properties of a chart object, where the color is defined by a red component **r**, a green component **g**, and a blue component **b**.

Syntax:

5 Functions in scripts and chart expressions

RGB (r, g, b)

Return data type: dual

Arguments:

Argument	Description
r, g, b	Red, green, and blue component values. A color component of 0 corresponds to no contribution and one of 255 to full contribution.



All arguments must be expressions that resolve to integers in the range 0 to 255.

If interpreting the numeric component and formatting it in hexadecimal notation, the values of the color components are easier to see. For example, light green has the number 4 278 255 360, which in hexadecimal notation is FF00FF00. The first two positions 'FF' (255) denote the **alpha** factor. In the functions **RGB** and **HSL**, this is always 'FF' (opaque). The next two positions '00' denote the amount of **red**, the next two positions 'FF' denote the amount of **green** and the final two positions '00' denote the amount of **blue**.

HSL

HSL() is used in expressions to set or evaluate the color properties of a chart object, where the color is defined by values of **hue**, **saturation**, and **luminosity** between 0 and 255.

Syntax:

HSL (hue, saturation, luminosity)

Return data type: dual

Arguments:

Argument	Description
hue, saturation, luminosity	hue, saturation, and luminosity component values. A value of 0 corresponds to no contribution and one of 255 to full contribution.



All arguments must be expressions that resolve to integers in the range 0 to 255.

If interpreting the numeric component and formatting it in hexadecimal notation, the values of the color components are easier to see. For example, light green has the number 4 286 080 100, which in hexadecimal notation is FF786464. The first two positions 'FF' (255) denote the **alpha** factor. In the functions **RGB** and **HSL**, this is always 'FF' (opaque). The next two positions '78' denote **hue** component, the next two positions '64' denote the **saturation**, and the final two positions '64' denote the **luminosity** component.

5.3 Conditional functions

The conditional functions all evaluate a condition and then return different answers depending on the condition value. The functions can be used in the data load script and in chart expressions.

Conditional functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

alt

The **alt** function returns the first of the parameters that has a valid number representation. If no such match is found, the last parameter will be returned. Any number of parameters can be used.

```
alt (case1[ , case2 , case3 , ...] , else)
```

class

The **class** function assigns the first parameter to a class interval. The result is a dual value with $a \leq x < b$ as the textual value, where a and b are the upper and lower limits of the bin, and the lower bound as numeric value.

```
class (expression, interval [ , label [ , offset ]])
```

if

The **if** function returns a value depending on whether the condition provided with the function evaluates as True or False.

```
if (condition , then , else)
```

match

The **match** function compares the first parameter with all the following ones and returns the number of expression that matches. The comparison is case sensitive.

```
match ( str, expr1 [ , expr2,...exprN ])
```

mixmatch

The **mixmatch** function compares the first parameter with all the following ones and returns the number of expression that matches. The comparison is case insensitive.

```
mixmatch ( str, expr1 [ , expr2,...exprN ])
```

pick

The **pick** function returns the n :th expression in the list.

```
pick (n, expr1[ , expr2,...exprN])
```

wildmatch

The **wildmatch** function compares the first parameter with all the following ones and returns the number of

5 Functions in scripts and chart expressions

expression that matches. It permits the use of wildcard characters (* and ?) in the comparison strings. The comparison is case insensitive.

```
wildmatch ( str, expr1 [ , expr2,...exprN ])
```

alt

The **alt** function returns the first of the parameters that has a valid number representation. If no such match is found, the last parameter will be returned. Any number of parameters can be used.

Syntax:

```
alt(case1[ , case2 , case3 , ...] , else)
```

The alt function is often used with number or date interpretation functions. This way, Qlik Sense can test different date formats in a prioritized order.

Examples and results:

Example	Result
<pre>alt(date#(dat , 'YYYY/MM/DD'), date#(dat , 'MM/DD/YYYY'), date#(dat , 'MM/DD/YY'), 'No valid date')</pre>	This expression will test if the field date contains a date according to any of the three specified date formats. If so, it will return a dual value containing the original string and a valid number representation of a date. If no match is found, the text 'No valid date' will be returned (without any valid number representation).

class

The **class** function assigns the first parameter to a class interval. The result is a dual value with $a \leq x < b$ as the textual value, where a and b are the upper and lower limits of the bin, and the lower bound as numeric value.

Syntax:

```
class(expression, interval [ , label [ , offset ]])
```

Arguments:

Argument	Description
interval	A number that specifies the bin width.
label	An arbitrary string that can replace the 'x' in the result text.
offset	A number that can be used as offset from the default starting point of the classification. The default starting point is normally 0.

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
<code>class(var,10) with var = 23</code>	returns '20<=x<30'
<code>class(var,5, 'value') with var = 23</code>	returns '20<= value <25'
<code>class(var,10, 'x',5) with var = 23</code>	returns '15<=x<25'

if

The **if** function returns a value depending on whether the condition provided with the function evaluates as True or False.

Syntax:

```
if( condition , then , else )
```

The if function has three parameters, *condition*, *then* and *else*, which are all expressions. The two other ones, *then* and *else*, can be of any type.

Arguments:

Argument	Description
condition	Expression that is interpreted logically.
then	Expression that can be of any type. If the <i>condition</i> is True, then the if function returns the value of the <i>then</i> expression.
else	Expression that can be of any type. If the <i>condition</i> is False, then the if function returns the value of the <i>else</i> expression.

Example:

```
if( Amount >= 0, 'OK', 'Alarm' )
```

match

The **match** function compares the first parameter with all the following ones and returns the number of expression that matches. The comparison is case sensitive.

Syntax:

```
match( str, expr1 [ , expr2, ...exprN ] )
```

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
<code>match(M, 'Jan', 'Feb', 'Mar')</code>	returns 2 if M = Feb returns 0 if M = Aprorjan

mixmatch

The **mixmatch** function compares the first parameter with all the following ones and returns the number of expression that matches. The comparison is case insensitive.

Syntax:

```
mixmatch( str, expr1 [ , expr2, ...exprN ])
```

Examples and results:

Example	Result
<code>mixmatch(M, 'Jan', 'Feb', 'Mar')</code>	returns 1 if M = jan

pick

The pick function returns the *n*:th expression in the list.

Syntax:

```
pick(n, expr1[ , expr2, ...exprN])
```

Arguments:

Argument	Description
<i>n</i>	<i>n</i> is an integer between 1 and N.

Examples and results:

Example	Result
<code>pick(N'A''B'4, , ,)</code>	returns 'B' if N = 2 returns 4 if N = 3

wildmatch

The **wildmatch** function compares the first parameter with all the following ones and returns the number of expression that matches. It permits the use of wildcard characters (* and ?) in the comparison strings. The comparison is case insensitive.

Syntax:

5 Functions in scripts and chart expressions

```
wildmatch( str, expr1 [ , expr2, ...exprN ])
```

Examples and results:

Example	Result
wildmatch(M, 'ja*', 'fe?', 'mar')	returns 1 if M = January returns 2 if M = fev

5.4 Counter functions

This section describes functions related to record counters during **LOAD** statement evaluation in the data load script. The only function that can be used in chart expressions is **RowNo()**.

Some counter functions do not have any parameters, but the trailing parentheses are however still required.

Counter functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

autonumber

This script function returns a unique integer value for each distinct evaluated value of *expression* encountered during the script execution. This function can be used e.g. for creating a compact memory representation of a complex key.

```
autonumber (expression[ , AutoID])
```

autonumberhash128

This script function calculates a 128-bit hash of the combined input expression values and returns a unique integer value for each distinct hash value encountered during the script execution. This function can be used for example for creating a compact memory representation of a complex key.

```
autonumberhash128 (expression {, expression})
```

autonumberhash256

This script function calculates a 256-bit hash of the combined input expression values and returns a unique integer value for each distinct hash value encountered during the script execution. This function can be used e.g. for creating a compact memory representation of a complex key.



This function is only available as a script function.

```
autonumberhash256 (expression {, expression})
```

fieldvaluecount

This script function returns the number of distinct values in a field. *fieldname* must be given as a string (for example a quoted literal).

5 Functions in scripts and chart expressions

fieldvaluecount (fieldname)

IterNo

This script function returns an integer indicating for which time one single record is evaluated in a **LOAD** statement with a **while** clause. The first iteration has number 1. The **IterNo** function is only meaningful if used together with a **while** clause.

IterNo ()

RecNo

This script functions returns an integer for the number of the currently read row of the current table. The first record is number 1.

RecNo ()

RowNo

This function returns an integer for the position of the current row in the resulting Qlik Sense internal table. The first row is number 1.

RowNo ()

autonumber

This script function returns a unique integer value for each distinct evaluated value of *expression* encountered during the script execution. This function can be used e.g. for creating a compact memory representation of a complex key.

Syntax:

autonumber(expression[, AutoID])

Arguments:

Argument	Description
AutoID	In order to create multiple counter instances if the autonumber function is used on different keys within the script, the optional parameter <i>AutoID</i> can be used for naming each counter.

Example 1:

```
autonumber( Region&Year&Month )
```

Example 2:

```
autonumber( Region&Year&Month, 'Ctr1' )
```

autonumberhash128

This script function calculates a 128-bit hash of the combined input expression values and returns a unique integer value for each distinct hash value encountered during the script execution. This function can be used for example for creating a compact memory representation of a complex key.

Syntax:

```
autonumberhash128 (expression {, expression})
```

Example:

```
autonumberhash128 ( Region, Year, Month )
```

autonumberhash256

This script function calculates a 256-bit hash of the combined input expression values and returns a unique integer value for each distinct hash value encountered during the script execution. This function can be used e.g. for creating a compact memory representation of a complex key.



This function is only available as a script function.

Syntax:

```
autonumberhash256 (expression {, expression})
```

Example:

```
Autonumberhash256 ( Region, Year, Month )
```

fieldvaluecount

This script function returns the number of distinct values in a field. *fieldname* must be given as a string (for example a quoted literal).

Syntax:

```
fieldvaluecount (fieldname)
```

Example:

```
let x = fieldvaluecount('Alfa');
```

IterNo

This script function returns an integer indicating for which time one single record is evaluated in a **LOAD** statement with a **while** clause. The first iteration has number 1. The **IterNo** function is only meaningful if used together with a **while** clause.

Syntax:

```
IterNo ( )
```

Examples and results:

Example	Result
<pre>LOAD StartDate, EndDate, IterNo() as DayWithinRange, Date(StartDate + IterNo() - 1) as Date while StartDate + IterNo() - 1 <= EndDate</pre>	This LOAD statement will generate one record per date within the range defined by StartDate and EndDate .

RecNo

This script functions returns an integer for the number of the currently read row of the current table. The first record is number 1.

Syntax:

```
RecNo ( )
```

RowNo

This function returns an integer for the position of the current row in the resulting Qlik Sense internal table. The first row is number 1.

Syntax:

```
RowNo ( [TOTAL] )
```

In contrast to **RecNo()**, which counts the records in the raw data table, the **RowNo()** function does not count records that are excluded by **where** clauses and is not reset when a raw data table is concatenated to another.



*If you use preceding load, that is, a number of stacked **LOAD** statements reading from the same table, you can only use **RowNo()** in the top **LOAD** statement. If you use **RowNo()** in subsequent **LOAD** statements, 0 is returned.*

Example: Data load script

Raw data tables:

Tab1.csv

A	B
1	aa
2	cc
3	ee

Tab2.csv

A	B
5	xx
4	yy
6	zz

QVTab:

```
LOAD *, RecNo( ), RowNo( ) from Tab1.csv where A<>2;  
LOAD *, RecNo( ), RowNo( ) from Tab2.csv where A<>5;
```

The resulting Qlik Sense internal table:

QVTab

A	B	RecNo()	RowNo()
1	aa	1	1
3	ee	3	2
4	yy	2	3
6	zz	3	4

RowNo - chart function

RowNo() returns the number of the current row within the current column segment in a table. For bitmap charts, **RowNo()** returns the number of the current row within the chart's straight table equivalent.

If the table or table equivalent has multiple vertical dimensions, the current column segment will include only rows with the same values as the current row in all dimension columns, except for the column showing the last dimension in the inter-field sort order.

Syntax:

```
RowNo ( [ TOTAL ] )
```

Return data type: integer

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

Limitations:

In contrast to `RecNo()`, which counts the records in the raw data table, the `RowNo()` function does not count records that are excluded by where clauses and is not reset when a raw data table is concatenated to another. The first row is number 1.

Examples and results:

Examples	Results
<code>if(RowNo()=1, 0, sum(Sales) / Above(sum(Sales)))</code>	The first row is number 1.

Example: Concatenating tables and counting rows when records are excluded

Tab1.csv:

```
A B
1 aa
2 cc
3 ee
```

Tab2.csv:

```
A B
5 xx
4 yy
6 zz
```

QVTab:

```
LOAD *, RecNo(), RowNo() from Tab1.csv where A<>2;
LOAD *, RecNo(), RowNo() from Tab2.csv where A<>5;
```

The resulting Qlik Sense internal table:

A	B	RecNo()	RowNo()
1	aa	1	1
3	ee	3	2
4	yy	2	3
6	zz	3	4

See also:

- ▢ [Above - chart function \(page 449\)](#)

5.5 Date and time functions

Qlik Sense date and time functions are used to transform and convert date and time values. All functions can be used in both the data load script and in chart expressions.

Functions are based on a date-time serial number that equals the number of days since Dec 30, 1899. The integer value represents the day and the fractional value represents the time of the day.

Qlik Sense uses the numerical value of the parameter, so a number is valid as a parameter also when it is not formatted as a date or a time. If the parameter lacks numeric value, e.g. is a string, then Qlik Sense attempts to interpret the string according to the date and time environment variables.

If the time format used in the parameter does not correspond to the one set in the environment variables, Qlik Sense will not be able to make a correct interpretation. To solve this problem, either change the settings or use an interpretation function.

In the below examples the default time and date formats hh:mm:ss and YYYY-MM-DD (ISO 8601) are assumed.

Date and time functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Integer expressions of time

second

This function returns an integer representing the second when the fraction of the **expression** is interpreted as a time according to the standard number interpretation.

```
second (expression)
```

minute

This function returns an integer representing the minute when the fraction of the **expression** is interpreted as a time according to the standard number interpretation.

```
minute (expression)
```

hour

This function returns an integer representing the hour when the fraction of the **expression** is interpreted as a time according to the standard number interpretation.

```
hour (expression)
```

5 Functions in scripts and chart expressions

day

This function returns an integer representing the day when the fraction of the **expression** is interpreted as a date according to the standard number interpretation.

```
day (expression)
```

week

This function returns an integer representing the week number according to ISO 8601. The week number is calculated from the date interpretation of the expression, according to the standard number interpretation.

```
week (expression)
```

month

This function returns a dual value with a month name as defined in the environment variable **MonthNames** and an integer between 1-12. The month is calculated from the date interpretation of the expression, according to the standard number interpretation.

```
month (expression)
```

year

This function returns an integer representing the year when the **expression** is interpreted as a date according to the standard number interpretation.

```
year (expression)
```

weekyear

This function returns the year to which the week number belongs according to ISO 8601. The week number ranges between 1 and approximately 52.

```
weekyear (expression)
```

weekday

This function returns a dual value with: A day name as defined in the environment variable **DayNames**. An integer between 0-6 corresponding to the nominal day of the week (0-6).

```
weekday (date)
```

Timestamp functions

now

This function returns a timestamp of the current time from the system clock.

```
now ([ timer_mode])
```

today

This function returns the current date from the system clock.

```
today ([timer_mode])
```

LocalTime

5 Functions in scripts and chart expressions

This function returns a timestamp of the current time from the system clock for a specified time zone.

```
localtime ([timezone [, ignoreDST ]])
```

Make functions

makedate

This function returns a date calculated from the year **YYYY**, the month **MM** and the day **DD**.

```
makedate (YYYY [ , MM [ , DD ] ])
```

makeweekdate

This function returns a date calculated from the year **YYYY**, the week **WW** and the day-of-week **D**.

```
makeweekdate (YYYY [ , WW [ , D ] ])
```

maketime

This function returns a time calculated from the hour **hh**, the minute **mm** the second **ss** with a fraction **fff** down to a millisecond.

```
maketime (hh [ , mm [ , ss [ .fff ] ] ])
```

Other date functions

AddMonths

This function returns the date occurring **n** months after **startdate** or, if **n** is negative, the date occurring **n** months before **startdate**.

```
addmonths (startdate, n , [ , mode])
```

AddYears

This function returns the date occurring **n** years after **startdate** or, if **n** is negative, the date occurring **n** years before **startdate**.

```
addmonths (startdate, n)
```

yeartodate

This function returns True if a **date** falls within the year to date, else False.

```
yeartodate (date [ , yearoffset [ , firstmonth [ , todaydate] ] ])
```

Timezone functions

timezone

This function returns the name of the current time zone, as defined in Windows.

```
timezone ( )
```

GMT

This function returns the current Greenwich Mean Time, as derived from the system clock and Windows time

5 Functions in scripts and chart expressions

settings.

```
GMT ( )
```

UTC

Returns the current Coordinated Universal Time.

```
UTC ( )
```

daylightsaving

Returns the current adjustment for daylight saving time, as defined in Windows.

```
daylightsaving ( )
```

converttolocaltime

Converts a UTC or GMT timestamp to local time as a dual value. The place can be any of a number of cities, places and time zones around the world.

```
converttolocaltime (timestamp [, place [, ignore_dst=false]])
```

Set time functions

setdateyear

This function returns a timestamp based on the input **timestamp** but with the year replaced with **year**.

```
setdateyear (timestamp, year)
```

setdateyearmonth

Returns a timestamp based on the input **timestamp** but with the year replaced with **year** and the month replaced with **month**

```
setdateyearmonth (timestamp, year, month)
```

inyear

This function returns True if **date** lies inside the year containing **basedate**.

```
inyear (date, basedate , shift [, first_month_of_year = 1])
```

In... functions

inyeartodate

This function returns True if **date** lies inside the part of year containing **basedate** up until and including the last millisecond of **basedate**.

```
inyeartodate (date, basedate , shift [, first_month_of_year = 1])
```

inquarter

This function returns True if **date** lies inside the quarter containing **basedate**.

```
inquarter (date, basedate , shift [, first_month_of_year = 1])
```

inquartertodate

5 Functions in scripts and chart expressions

This function returns True if **date** lies inside the part of the quarter containing **basedate** up until and including the last millisecond of **basedate**.

```
inquartertodate (date, basedate , shift [, first_month_of_year = 1])
```

inmonth

This function returns True if **date** lies inside the month containing **basedate**.

```
inmonth (date, basedate , shift)
```

inmonthtodate

Returns True if **date** lies inside the part of month containing **basedate** up until and including the last millisecond of **basedate**.

```
inmonthtodate (date, basedate , shift)
```

inmonths

Returns True if **date** lies inside the n month shift (aligned from January 1st) containing **basedate**.

```
inmonths (n, date, basedate , shift [, first_month_of_year = 1])
```

inmonthstodate

This function returns True if **date** lies inside the n month shift (aligned from January 1st) containing **basedate**.

```
inmonthstodate (n, date, basedate , shift [, first_month_of_year = 1])
```

inweek

This function returns True if **date** lies inside the week containing **basedate**.

```
inweek (date, basedate , shift [, weekstart])
```

inweektodate

This function returns True if **date** lies inside the part of week containing **basedate** up until and including the last millisecond of **basedate**.

```
inweektodate (date, basedate , shift [, weekstart])
```

inlunarweek

This function returns True if **date** lies inside the lunar week (consecutive 7 day periods starting on January 1st each year) containing **basedate**.

```
inlunarweek (date, basedate , shift [, weekstart])
```

inlunarweektodate

This function returns True if **date** lies inside the part of lunar week (consecutive 7 day periods starting on January 1st each year) containing **basedate** up until and including the last millisecond of **basedate**.

```
inlunarweektodate (date, basedate , shift [, weekstart])
```

inday

5 Functions in scripts and chart expressions

This function returns True if **timestamp** lies inside the day containing **basetimestamp**.

```
inday (timestamp, basetimestamp , shift [, daystart])
```

indaytotime

This function returns True if **timestamp** lies inside the part of day containing **basetimestamp** up until and including the exact millisecond of **basetimestamp**.

```
indaytotime (timestamp, basetimestamp , shift [, daystart])
```

Start ... end functions

yearstart

This function returns a value corresponding to a timestamp of the first millisecond of the first day of the year containing **date**. The default output format will be the **DateFormat** set in the script.

```
yearstart ( date [, shift = 0 [, first_month_of_year = 1]])
```

yearend

This function returns a value corresponding to a timestamp of the last millisecond of the last day of the year containing **date**. The default output format will be the **DateFormat** set in the script.

```
yearend ( date [, shift = 0 [, first_month_of_year = 1]])
```

yearname

This function returns a four-digit year as display value with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the year containing **date**.

```
yearname (date [, shift = 0 [, first_month_of_year = 1]] )
```

quarterstart

This function returns a value corresponding to a timestamp of the first millisecond of the quarter containing **date**. The default output format will be the **DateFormat** set in the script.

```
quarterstart (date [, shift = 0 [, first_month_of_year = 1]])
```

quarterend

This function returns a value corresponding to a timestamp of the last millisecond of the quarter containing **date**. The default output format will be the **DateFormat** set in the script.

```
quarterend (date [, shift = 0 [, first_month_of_year = 1]])
```

quartername

This function returns a display value showing the months of the quarter (formatted according to the **MonthNames** script variable) and year with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the quarter.

```
quartername (date [, shift = 0 [, first_month_of_year = 1]])
```

monthstart

5 Functions in scripts and chart expressions

This function returns a value corresponding to a timestamp of the first millisecond of the first day of the month containing **date**. The default output format will be the **DateFormat** set in the script.

```
monthstart (date [, shift = 0])
```

monthend

This function returns a value corresponding to a timestamp of the last millisecond of the last day of the month containing **date**. The default output format will be the **DateFormat** set in the script.

```
monthend (date [, shift = 0])
```

monthname

This function returns a display value showing the month (formatted according to the **MonthNames** script variable) and year with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the month.

```
monthname (date [, shift = 0])
```

monthsstart

This function returns a value corresponding to a timestamp of the first millisecond of the *n* month period (starting from January 1st) containing **date**. The default output format will be the **DateFormat** set in the script.

```
monthsstart (n, date [, shift = 0 [, first_month_of_year = 1]])
```

monthsend

This function returns a value corresponding to a timestamp of the last millisecond of the *n* month period (starting from January 1st) containing **date**. The default output format will be the **DateFormat** set in the script.

```
monthsend (n, date [, shift = 0 [, first_month_of_year = 1]])
```

monthsname

This function returns a display value showing the months of the period (formatted according to the **MonthNames** script variable) and year with an underlying numeric value corresponding to a timestamp of the first millisecond of the *n* month period (starting from January 1st) containing **date**.

```
monthsname (n, date [, shift = 0 [, first_month_of_year = 1]])
```

weekstart

This function returns a value corresponding to a timestamp of the first millisecond of the first day (Monday) of the calendar week containing **date**. The default output format is the **DateFormat** set in the script.

```
weekstart (date [, shift = 0 [, weekoffset = 0]])
```

weekend

This function returns a value corresponding to a timestamp of the last millisecond of the last day (Sunday) of the calendar week containing **date**. The default output format will be the **DateFormat** set in the script.

5 Functions in scripts and chart expressions

```
weekend (date [, shift = 0 [,weekoffset = 0]])
```

weekname

This function returns a value showing the year and week number with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the week containing **date**.

```
weekname (date [, shift = 0 [,weekoffset = 0]])
```

lunarweekstart

This function returns a value corresponding to a timestamp of the first millisecond of the lunar week (consecutive 7 day periods starting on January 1st each year) containing **date**. The default output format will be the **DateFormat** set in the script.

```
lunarweekstart (date [, shift = 0 [,weekoffset = 0]])
```

lunarweekend

This function returns a value corresponding to a timestamp of the last millisecond of the lunar week (consecutive 7 day periods starting on January 1st each year) containing date. The default output format will be the **DateFormat** set in the script.

```
lunarweekend (date [, shift = 0 [,weekoffset = 0]])
```

lunarweekname

This function returns a display value showing the year and week number with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the lunar week (consecutive 7 day periods starting on January 1st each year)containing date.

```
lunarweekname (date [, shift = 0 [,weekoffset = 0]])
```

daystart

This function returns a value corresponding to a timestamp with the first millisecond of the day contained in **timestamp**. The default output format will be the **TimestampFormat** set in the script.

```
daystart (timestamp [, shift = 0 [, dayoffset = 0]])
```

dayend

This function returns a value corresponding to a timestamp of the last millisecond of the day. The default output format will be the **TimestampFormat** set in the script.

```
dayend (timestamp [, shift = 0 [, dayoffset = 0]])
```

dayname

This function returns a value showing the date with an underlying numeric value corresponding to a timestamp of the first millisecond of the day containing **timestamp**.

```
dayname (timestamp [, shift = 0 [, dayoffset = 0]])
```

Day numbering functions

age

5 Functions in scripts and chart expressions

Returns the age at the time of **timestamp** (in completed years) of somebody born on **date_of_birth**.

```
age (timestamp, date_of_birth)
```

networkdays

Returns the number of working days (Monday-Friday) between and including **start_date** and **end_date** taking into account any optionally listed **holiday**. All parameters should be valid dates or timestamps.

```
networkdays (start:date, end_date {, holiday})
```

firstworkdate

Returns the latest starting date to achieve **no_of_workdays** (Monday-Friday) ending no later than **end_date** taking into account any optionally listed holidays. **end_date** and **holiday** should be valid dates or timestamps.

```
firstworkdate (end_date, no_of_workdays {, holiday} )
```

lastworkdate

Returns the earliest ending date to achieve **no_of_workdays** (Monday-Friday) if starting at **start_date** taking into account any optionally listed **holiday**. **start_date** and **holiday** should be valid dates or timestamps.

```
lastworkdate (start_date, no_of_workdays {, holiday})
```

daynumberofyear

Returns the day number of the year according to a timestamp of the first millisecond of the first day of the year containing **date**.

```
daynumberofyear (date[,firstmonth])
```

daynumberofquarter

Returns the day number of the quarter according to a timestamp of the first millisecond of the first day of the quarter containing **date**.

```
daynumberofquarter (date[,firstmonth])
```

addmonths

This function returns the date occurring **n** months after **startdate** or, if **n** is negative, the date occurring **n** months before **startdate**.

Syntax:

```
AddMonths (startdate, n , [ , mode])
```

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
startdate	The start date as a time stamp, for example '2012-10-12'.
n	Number of months as a positive or negative integer.
mode	mode specifies if the month is added relative to the beginning of the month or relative to the end of the month. If the input date is the 28th or above and mode is set to 1, the function will return a date which is the same distance from the end of the month as the input date. Default mode is 0.

Examples and results:

Example	Result
addmonths ('2003-01-29', 3)	returns '2003-04-29'
addmonths ('2003-01-29', 3, 0)	returns '2003-04-29'
addmonths ('2003-01-29', 3, 1)	returns '2003-04-28'
addmonths ('2003-01-29', 1, 0)	returns '2003-02-28'
addmonths ('2003-01-29', 1, 1)	returns '2003-02-26'
addmonths ('2003-02-28', 1, 0)	returns '2003-03-28'
addmonths ('2003-02-28', 1, 1)	returns '2003-03-31'

addyears

This function returns the date occurring **n** years after **startdate** or, if **n** is negative, the date occurring **n** years before **startdate**.

Syntax:

```
AddYears (startdate, n)
```

Arguments:

Argument	Description
startdate	The start date as a time stamp, for example '2012-10-12'.
n	Number of years as a positive or negative integer.

Examples and results:

Example	Result
addyears ('2010-01-29', 3)	returns '2013-01-29'
addyears ('2010-01-29', -1)	returns '2009-01-29'

age

Returns the age at the time of **timestamp** (in completed years) of somebody born on **date_of_birth**.

Syntax:

```
age(timestamp, date_of_birth)
```

Examples and results:

Example	Result
age('2007-01-25', '2005-10-29')	Returns 1
age('2007-10-29', '2005-10-29')	Returns 2

converttolocaltime

Converts a UTC or GMT timestamp to local time as a dual value. The place can be any of a number of cities, places and time zones around the world.

Syntax:

```
ConvertToLocalTime(timestamp [, place [, ignore_dst=false]])
```

Also GMT, GMT-01:00, GMT+04:00 etc. are valid places.

The resulting time is adjusted for daylight savings time, unless the third parameter is set to True().

Valid places and time zones

Abu Dhabi	Central America	Kabul	Nairobi	Sydney
Adelaide	Central Time (US & Canada)	Kamchatka	New Caledonia	Taipei
Alaska	Chennai	Karachi	New Delhi	Tallinn
Amsterdam	Chihuahua	Kathmandu	Newfoundland	Tashkent
Arizona	Chongqing	Kolkata	Novosibirsk	Tbilisi
Astana	Copenhagen	Krasnoyarsk	Nuku'alofa	Tehran
Athens	Darwin	Kuala Lumpur	Osaka	Tokyo
Atlantic Time (Canada)	Dhaka	Kuwait	Pacific Time (US & Canada)	Urumqi
Auckland	Eastern Time (US & Canada)	Kyiv	Paris	Warsaw

5 Functions in scripts and chart expressions

Valid places and time zones

	Canada)			
Azores	Edinburgh	La Paz	Perth	Wellington
Baghdad	Ekaterinburg	Lima	Port Moresby	West Central Africa
Baku	Fiji	Lisbon	Prague	Vienna
Bangkok	Georgetown	Ljubljana	Pretoria	Vilnius
Beijing	Greenland	London	Quito	Vladivostok
Belgrade	Greenwich Mean Time : Dublin	Madrid	Riga	Volgograd
Berlin	Guadalajara	Magadan	Riyadh	Yakutsk
Bern	Guam	Mazatlan	Rome	Yerevan
Bogota	Hanoi	Melbourne	Samoa	Zagreb
Brasilia	Harare	Mexico City	Santiago	
Bratislava	Hawaii	Mid-Atlantic	Sapporo	
Brisbane	Helsinki	Minsk	Sarajevo	
Brussels	Hobart	Karachi	Saskatchewan	
Bucharest	Hong Kong	Kathmandu	Seoul	
Budapest	Indiana (East)	Kolkata	Singapore	
Buenos Aires	International Date Line West	Monrovia	Skopje	
Cairo	Irkutsk	Monterrey	Sofia	
Canberra	Islamabad	Moscow	Solomon Is.	
Cape Verde Is.	Istanbul	Mountain Time (US & Canada)	Sri Jayawardenepura	
Caracas	Jakarta	Mumbai	St. Petersburg	
Casablanca	Jerusalem	Muscat	Stockholm	

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
<code>convertToLocalTime('2007-11-10 23:59:00', 'Paris')</code>	Returns '2007-11-11 00:59:00' and the corresponding internal timestamp representation.
<code>convertToLocalTime(UTC(), 'GMT-05:00')</code>	Returns the time for the North American east coast, e.g. New York.

day

This function returns an integer representing the day when the fraction of the **expression** is interpreted as a date according to the standard number interpretation.

Syntax:

```
day (expression)
```

Examples and results:

Example	Result
<code>day('1971-10-12')</code>	returns 30

dayend

This function returns a value corresponding to a timestamp of the last millisecond of the day. The default output format will be the **TimestampFormat** set in the script.

Syntax:

```
DayEnd(timestamp [, shift = 0 [, dayoffset = 0]])
```

Arguments:

Argument	Description
timestamp	The timestamp to evaluate.
shift	shift is an integer, where the value 0 indicates the day which contains timestamp . Negative values in shift indicate preceding days and positive values indicate succeeding days.
dayoffset	If you want to work with days not starting midnight, indicate an offset in fraction of a day in dayoffset , e.g 0.125 to denote 3 AM.

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
<code>dayend ('2006-01-25 16:45')</code>	Returns '2006-01-25 23:59:59' with an underlying numeric value corresponding to '2006-01-25 23:59:59.999'
<code>dayend ('2006-01-25 16:45', -1)</code>	Returns '2006-01-24 23:59:59' with an underlying numeric value corresponding to '2006-01-24 23:59:59.999'
<code>dayend ('2006-01-25 16:45', 0, 0.5)</code>	Returns '2006-01-26 11:59:59' with an underlying numeric value corresponding to '2006-01-26 11:59:59.999'

daylightsaving

Returns the current adjustment for daylight saving time, as defined in Windows.

Syntax:

```
DaylightSaving( )
```

Example:

```
daylightsaving( )
```

dayname

This function returns a value showing the date with an underlying numeric value corresponding to a timestamp of the first millisecond of the day containing **timestamp**.

Syntax:

```
DayName (timestamp [, shift = 0 [, dayoffset = 0]])
```

Arguments:

Argument	Description
timestamp	The timestamp to evaluate.
shift	shift is an integer, where the value 0 indicates the day which contains timestamp . Negative values in shift indicate preceding days and positive values indicate succeeding days.
dayoffset	If you want to work with days not starting midnight, indicate an offset in fraction of a day in dayoffset , e.g 0.125 to denote 3 AM.

Examples and results:

Example	Result
<code>dayname ('2006-01-25 16:45')</code>	Returns '2006-01-25' with an underlying numeric value

5 Functions in scripts and chart expressions

Example	Result
	corresponding to '2006-01-25 00:00:00.000'
dayname ('2006-01-25 16:45', -1)	Returns '2006-01-24' with an underlying numeric value corresponding to '2006-01-24 00:00:00.000'
dayname ('2006-01-25 16:45', 0, 0.5)	Returns '2006-01-25' with an underlying numeric value corresponding to '2006-01-25 12:00:00.000'

daynumberofquarter

Returns the day number of the quarter according to a timestamp of the first millisecond of the first day of the quarter containing **date**.

Syntax:

```
DayNumberOfQuarter (date[, firstmonth])
```

The function always uses years based on 366 days.

Arguments:

Argument	Description
date	The date to evaluate.
firstmonth	By specifying a firstmonth between 1 and 12 (1 if omitted), the beginning of the year may be moved forward to the first day of any month. If you e.g. want to work with a fiscal year starting March 1, specify firstmonth = 3.

Examples and results:

Example	Result
DayNumberOfQuarter(Date)	Returns the day number of the quarter counted from the first day of the first quarter.
DayNumberOfQuarter(Date, 3)	Returns the day number of the quarter counted from the first of March.

daynumberofyear

Returns the day number of the year according to a timestamp of the first millisecond of the first day of the year containing **date**.

Syntax:

```
DayNumberOfYear (date[, firstmonth])
```

5 Functions in scripts and chart expressions

The function always uses years based on 366 days.

Arguments:

Argument	Description
date	The date to evaluate.
firstmonth	By specifying a firstmonth between 1 and 12 (1 if omitted), the beginning of the year may be moved forward to the first day of any month. If you e.g. want to work with a fiscal year starting March 1, specify firstmonth = 3.

Examples and results:

Example	Result
DayNumberOfYear(date)	Returns the day number counted from the first of the year.
DayNumberOfYear(date, 3)	Returns the number of the day as counted from the first of March.

daystart

This function returns a value corresponding to a timestamp with the first millisecond of the day contained in **timestamp**. The default output format will be the **TimestampFormat** set in the script.

Syntax:

```
DayStart(timestamp [, shift = 0 [, dayoffset = 0]])
```

Arguments:

Argument	Description
timestamp	The timestamp to evaluate.
shift	shift is an integer, where the value 0 indicates the day which contains timestamp . Negative values in shift indicate preceding days and positive values indicate succeeding days.
dayoffset	If you want to work with days not starting midnight, indicate an offset in fraction of a day in dayoffset , e.g 0.125 to denote 3 AM.

Examples and results:

Example	Result
daystart ('2006-01-25 16:45')	Returns '2006-01-25 00:00:00' with an underlying numeric value corresponding to '2006-01-25 00:00:00.000'

5 Functions in scripts and chart expressions

Example	Result
<code>daystart ('2006-01-25 16:45', -1)</code>	Returns '2006-01-24 00:00:00' with an underlying numeric value corresponding to '2006-01-24 00:00:00.000'
<code>daystart ('2006-01-25 16:45', 0, 0.5)</code>	Returns '2006-01-25 12:00:00' with an underlying numeric value corresponding to '2006-01-25 12:00:00.000'

firstworkdate

Returns the latest starting date to achieve **no_of_workdays** (Monday-Friday) ending no later than **end_date** taking into account any optionally listed holidays. **end_date** and **holiday** should be valid dates or timestamps.

Syntax:

```
firstworkdate(end_date, no_of_workdays {, holiday} )
```

Examples and results:

Example	Result
<code>firstworkdate ('2007-03-01', 9)</code>	Returns '2007-02-19'
<code>firstworkdate ('2006-12-31', 8, '2006-12-25', '2006-12-26')</code>	Returns '2006-12-18'

GMT

This function returns the current Greenwich Mean Time, as derived from the system clock and Windows time settings.

Syntax:

```
GMT ( )
```

Example:

```
gmt ( )
```

hour

This function returns an integer representing the hour when the fraction of the **expression** is interpreted as a time according to the standard number interpretation.

Syntax:

```
hour (expression)
```

5 Functions in scripts and chart expressions

Examples and results:

Example	Result
hour('09:14:36')	returns 9
hour('0.5555')	returns 13 (Because 0.5555 = 13:19:55)

inday

This function returns True if **timestamp** lies inside the day containing **basetimestamp**.

Syntax:

```
InDay (timestamp, basetimestamp , shift [, daystart])
```

Arguments:

Argument	Description
timestamp	The date and time that you want to compare with basetimestamp .
basetimestamp	Date and time that is used to evaluate the timestamp.
shift	The day can be offset by shift . shift is an integer, where the value 0 indicates the day which contains basetimestamp . Negative values in shift indicate preceding days and positive values indicate succeeding days.
daystart	If you want to work with days not starting midnight, indicate an offset in fraction of a day in daystart , e.g 0.125 to denote 3 AM.

Examples and results:

Example	Result
inday ('2006-01-12 12:23', '2006-01-12 00:00', 0)	Returns True
inday ('2006-01-12 12:23', '2006-01-13 00:00', 0)	Returns False
inday ('2006-01-12 12:23', '2006-01-12 00:00', -1)	Returns False
inday ('2006-01-11 12:23', '2006-01-12 00:00', -1)	Returns True
inday ('2006-01-12 12:23', '2006-01-12 00:00', 0, 0.5)	Returns False
inday ('2006-01-12 11:23', '2006-01-12 00:00', 0, 0.5)	Returns True

indaytotime

This function returns True if **timestamp** lies inside the part of day containing **basetimestamp** up until and including the exact millisecond of **basetimestamp**.

Syntax:

```
InDayToTime (timestamp, basetimestamp , shift [, daystart])
```

5 Functions in scripts and chart expressions

Arguments:

Argument	Description
timestamp	The date and time that you want to compare with basetimestamp .
basetimestamp	Date and time that is used to evaluate the timestamp.
shift	The day can be offset by shift . shift is an integer, where the value 0 indicates the day which contains basetimestamp . Negative values in shift indicate preceding days and positive values indicate succeeding days.
daystart	If you want to work with days not starting midnight, indicate an offset in fraction of a day in daystart , e.g 0.125 to denote 3 AM.

Examples and results:

Example	Result
<code>indaytotime ('2006-01-12 12:23', '2006-01-12 23:59', 0)</code>	Returns True
<code>indaytotime ('2006-01-12 12:23', '2006-01-12 00:00', 0)</code>	Returns False
<code>indaytotime ('2006-01-11 12:23', '2006-01-12 23:59', -1)</code>	Returns True

inlunarweek

This function returns True if **date** lies inside the lunar week (consecutive 7 day periods starting on January 1st each year) containing **basedate**.

Syntax:

```
InLunarWeek (date, basedate , shift [, weekstart])
```

Arguments:

Argument	Description
date	The date that you want to compare with basedate .
basedate	Date that is used to evaluate the lunar week.
shift	The lunar week can be offset by shift . shift is an integer, where the value 0 indicates the lunar week which contains basedate . Negative values in shift indicate preceding lunar weeks and positive values indicate succeeding lunar weeks.
weekstart	If you want to work with an offset for the start of the lunar weeks, indicate an offset in days in weekstart . This may be given as a real number indicating days and/or fractions of a day.

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
<code>inLunarWeek ('2006-01-12', '2006-01-14', 0)</code>	Returns True
<code>inLunarWeek ('2006-01-12', '2006-01-20', 0)</code>	Returns False
<code>inLunarWeek ('2006-01-12', '2006-01-14', -1)</code>	Returns False
<code>inLunarWeek ('2006-01-07', '2006-01-14', -1)</code>	Returns True
<code>inLunarWeek ('2006-01-11', '2006-01-08', 0, 3)</code>	Returns False

inLunarWeekToDate

This function returns True if **date** lies inside the part of lunar week (consecutive 7 day periods starting on January 1st each year) containing **basedate** up until and including the last millisecond of **basedate**.

Syntax:

```
InLunarWeekToDate (date, basedate , shift [, weekstart])
```

Arguments:

Argument	Description
date	The date that you want to compare with basedate .
basedate	Date that is used to evaluate the lunar week.
shift	The lunar week can be offset by shift . shift is an integer, where the value 0 indicates the lunar week which contains basedate . Negative values in shift indicate preceding lunar weeks and positive values indicate succeeding lunar weeks.
weekstart	If you want to work with an offset for the start of the lunar weeks, indicate an offset in days in weekstart . This may be given as a real number indicating days and/or fractions of a day.

Examples and results:

Example	Result
<code>inLunarWeekToDate ('2006-01-12', '2006-01-12', 0)</code>	Returns True
<code>inLunarWeekToDate ('2006-01-12', '2006-01-11', 0)</code>	Returns False
<code>inLunarWeekToDate ('2006-01-12', '2006-01-05', 1)</code>	Returns True

inMonth

This function returns True if **date** lies inside the month containing **basedate**.

Syntax:

```
InMonth (date, basedate , shift)
```

5 Functions in scripts and chart expressions

Arguments:

Argument	Description
date	The date that you want to compare with basedate .
basedate	Date that is used to evaluate the month.
shift	The month can be offset by shift . shift is an integer, where the value 0 indicates the month which contains basedate . Negative values in shift indicate preceding months and positive values indicate succeeding months.

Examples and results:

Example	Result
<code>inmonth ('2006-01-25', '2006-01-01', 0)</code>	Returns True
<code>inmonth('2006-01-25', '2006-04-01', 0)</code>	Returns False
<code>inmonth ('2006-01-25', '2006-01-01', -1)</code>	Returns False
<code>inmonth ('2005-12-25', '2006-01-01', -1)</code>	Returns True

inmonths

Returns True if **date** lies inside the n month shift (aligned from January 1st) containing **basedate**.

Syntax:

```
InMonths (n, date, basedate , shift [, first_month_of_year = 1])
```

Arguments:

Argument	Description
n	An integer that must be (1), 2, (3), 4 or 6.
date	The date that you want to compare with basedate .
basedate	Date that is used to evaluate the period.
shift	The period can be offset by shift . shift is an integer, where the value 0 indicates the period which contains basedate . Negative values in shift indicate preceding periods and positive values indicate succeeding periods.
first_ month_ of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
<code>inmonthstodate (4, '2006-01-25', '2006-04-25', 0)</code>	Returns True
<code>inmonthstodate (4, '2006-04-25', '2006-04-24', 0)</code>	Returns False
<code>inmonthstodate (4, '2005-11-25', '2006-02-01', -1)</code>	Returns True

inmonthstodate

This function returns True if **date** lies inside the **n** month shift (aligned from January 1st) containing **basedate**.

Syntax:

```
InMonths (n, date, basedate , shift [, first_month_of_year = 1])
```

Arguments:

Argument	Description
n	An integer that must be (1), 2, (3), 4 or 6.
date	The date that you want to compare with basedate .
basedate	Date that is used to evaluate the period.
shift	The period can be offset by shift . shift is an integer, where the value 0 indicates the period which contains basedate . Negative values in shift indicate preceding periods and positive values indicate succeeding periods.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

Examples and results:

Example	Result
<code>inmonthstodate (4, '2006-01-25', '2006-04-25', 0)</code>	Returns True
<code>inmonthstodate (4, '2006-04-25', '2006-04-24', 0)</code>	Returns False
<code>inmonthstodate (4, '2005-11-25', '2006-02-01', -1)</code>	Returns True

inmonthtodate

Returns True if **date** lies inside the part of month containing **basedate** up until and including the last millisecond of **basedate**.

Syntax:

```
InMonthToDate (date, basedate , shift)
```

5 Functions in scripts and chart expressions

Arguments:

Argument	Description
date	The date that you want to compare with basedate .
basedate	Date that is used to evaluate the month.
shift	The month can be offset by shift . shift is an integer, where the value 0 indicates the month which contains basedate . Negative values in shift indicate preceding months and positive values indicate succeeding months.

Examples and results:

Example	Result
<code>inmonthtodate ('2006-01-25', '2006-01-25', 0)</code>	Returns True
<code>inmonthtodate ('2006-01-25', '2006-01-24', 0)</code>	Returns False
<code>inmonthtodate ('2006-01-25', '2006-02-28', -1)</code>	Returns True

inquarter

This function returns True if **date** lies inside the quarter containing **basedate**.

Syntax:

```
InQuarter (date, basedate , shift [, first_month_of_year = 1])
```

Arguments:

Argument	Description
date	The date that you want to compare with basedate .
basedate	Date that is used to evaluate the quarter.
shift	The quarter can be offset by shift . shift is an integer, where the value 0 indicates the quarter which contains basedate . Negative values in shift indicate preceding quarters and positive values indicate succeeding quarters.
first_ month_ of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
<code>inquarter ('2006-01-25', '2006-01-01', 0)</code>	Returns True
<code>inquarter ('2006-01-25', '2006-04-01', 0)</code>	Returns False
<code>inquarter ('2006-01-25', '2006-01-01', -1)</code>	Returns False
<code>inquarter ('2005-12-25', '2006-01-01', -1)</code>	Returns True
<code>inquarter ('2006-01-25', '2006-03-01', 0, 3)</code>	Returns False
<code>inquarter ('2006-03-25', '2006-03-01', 0, 3)</code>	Returns True

inquartertodate

This function returns True if **date** lies inside the part of the quarter containing **basedate** up until and including the last millisecond of **basedate**.

Syntax:

```
InQuarterToDate (date, basedate , shift [, first_month_of_year = 1])
```

Arguments:

Argument	Description
date	The date that you want to compare with basedate .
basedate	Date that is used to evaluate the quarter.
shift	The quarter can be offset by shift . shift is an integer, where the value 0 indicates the quarter which contains basedate . Negative values in shift indicate preceding quarters and positive values indicate succeeding quarters.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

Examples and results:

Example	Result
<code>inquartertodate ('2006-01-25', '2006-01-25', 0)</code>	Returns True
<code>inquartertodate ('2006-01-25', '2006-01-24', 0)</code>	Returns False
<code>inquartertodate ('2005-12-25', '2006-02-01', -1)</code>	Returns True

inweek

This function returns True if **date** lies inside the week containing **basedate**.

Syntax:

5 Functions in scripts and chart expressions

InWeek (date, basedate , shift [, weekstart])

Arguments:

Argument	Description
date	The date that you want to compare with basedate .
basedate	Date that is used to evaluate the week.
shift	The week can be offset by shift . shift is an integer, where the value 0 indicates the week which contains basedate . Negative values in shift indicate preceding weeks and positive values indicate succeeding weeks.
weekstart	If you want to work with weeks not starting midnight between Sunday and Monday, indicate an offset in days in weekstart . This may be given as a real number indicating days and/or fractions of a day.

Examples and results:

Example	Result
<code>inweek ('2006-01-12', '2006-01-14', 0)</code>	Returns True
<code>inweek ('2006-01-12', '2006-01-20', 0)</code>	Returns False
<code>inweek ('2006-01-12', '2006-01-14', -1)</code>	Returns False
<code>inweek ('2006-01-07', '2006-01-14', -1)</code>	Returns True
<code>inweek ('2006-01-12', '2006-01-09', 0, 3)</code>	Returns False

inweektodate

This function returns True if **date** lies inside the part of week containing **basedate** up until and including the last millisecond of **basedate**.

Syntax:

InWeekToDate (date, basedate , shift [, weekstart])

Arguments:

Argument	Description
date	The date that you want to compare with basedate .
basedate	Date that is used to evaluate the week.
shift	The week can be offset by shift . shift is an integer, where the value 0 indicates the week which contains basedate . Negative values in shift indicate preceding weeks and positive values indicate succeeding weeks.

5 Functions in scripts and chart expressions

Argument	Description
weekstart	If you want to work with weeks not starting midnight between Sunday and Monday, indicate an offset in days in weekstart . This may be given as a real number indicating days and/or fractions of a day.

Examples and results:

Example	Result
<code>inweektoday ('2006-01-12', '2006-01-12', 0)</code>	Returns True
<code>inweektoday ('2006-01-12', '2006-01-11', 0)</code>	Returns False
<code>inweektoday ('2006-01-12', '2006-01-05', -1)</code>	Returns False

inyear

This function returns True if **date** lies inside the year containing **basedate**.

Syntax:

```
InYear (date, basedate , shift [, first_month_of_year = 1])
```

Arguments:

Argument	Description
date	The date that you want to compare with basedate .
basedate	Date that is used to evaluate the year.
shift	The year can be offset by shift . shift is an integer, where the value 0 indicates the year which contains basedate . Negative values in shift indicate preceding years and positive values indicate succeeding years.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

Examples and results:

Example	Result
<code>inyear ('2006-01-25', '2006-01-01', 0)</code>	Returns True
<code>inyear ('2005-01-25', '2006-01-01', 0)</code>	Returns False
<code>inyear ('2006-01-25', '2006-01-01', -1)</code>	Returns False
<code>inyear ('2005-01-25', '2006-01-01', -1)</code>	Returns True

5 Functions in scripts and chart expressions

Example	Result
<code>inyear ('2006-01-25', '2006-07-01', 0, 3)</code>	Returns False
<code>inyear ('2006-03-25', '2006-07-01', 0, 3)</code>	Returns True

inyeartodate

This function returns True if **date** lies inside the part of year containing **basedate** up until and including the last millisecond of **basedate**.

Syntax:

```
InYearToDate (date, basedate , shift [, first_month_of_year = 1])
```

Argument	Description
date	The date that you want to compare with basedate .
basedate	Date that is used to evaluate the year.
shift	The year can be offset by shift . shift is an integer, where the value 0 indicates the year which contains basedate . Negative values in shift indicate preceding years and positive values indicate succeeding years.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

Examples and results:

Example	Result
<code>inyeartodate ('2006-01-25', '2006-02-01', 0)</code>	Returns True
<code>inyear ('2005-01-25', '2006-01-01', 0)</code>	Returns False
<code>inyear ('2005-01-25', '2006-02-01', -1)</code>	Returns True

lastworkdate

Returns the earliest ending date to achieve **no_of_workdays** (Monday-Friday) if starting at **start_date** taking into account any optionally listed **holiday**. **start_date** and **holiday** should be valid dates or timestamps.

Syntax:

```
lastworkdate (start_date, no_of_workdays {, holiday})
```

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
<code>lastworkdate ('2007-02-19', 9)</code>	Returns '2007-03-01'
<code>lastworkdate ('2006-12-18', 8, '2006-12-25', '2006-12-26')</code>	Returns '2006-12-29'

localtime

This function returns a timestamp of the current time from the system clock for a specified time zone.

Syntax:

```
LocalTime([timezone [, ignoreDST ]])
```

Arguments:

Argument	Description
timezone	The timezone is specified as a string containing any of the geographical places listed under Time Zone in the Windows Control Panel for Date and Time or as a string in the form 'GMT+hh:mm'. If no time zone is specified the local time will be returned.
ignoreDST	If ignoreDST is -1 (True) daylight savings time will be ignored.

Examples:

```
localtime ('Paris')  
localtime ('GMT+01:00')  
localtime ('Paris',-1)  
localtime()
```

lunarweekend

This function returns a value corresponding to a timestamp of the last millisecond of the lunar week (consecutive 7 day periods starting on January 1st each year) containing date. The default output format will be the **DateFormat** set in the script.

Syntax:

```
LunarweekEnd(date [, shift = 0 [,weekoffset = 0]])
```

Arguments:

Argument	Description
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the lunar week which contains date .

5 Functions in scripts and chart expressions

Argument	Description
	Negative values in shift indicate preceding lunar weeks and positive values indicate succeeding lunar weeks.
weekoffset	If you want to work with an offset for the start of the lunar weeks, indicate an offset in days in of weekoffset . This may be given as a real number indicating days and/or fractions of a day.

Examples and results:

Example	Result
<code>lunarweekend ('2006-01-12')</code>	Returns '2006-01-14' with an underlying numeric value corresponding to '2006-01-14 23:59:59.999'
<code>lunarweekend ('2006-01-12', -1)</code>	Returns '2006-01-07' with an underlying numeric value corresponding to '2006-01-07 23:59:59.999'
<code>lunarweekend ('2006-01-12', 0, 1)</code>	Returns '2006-01-15' with an underlying numeric value corresponding to '2006-01-15 23:59:59.999'

lunarweekname

This function returns a display value showing the year and week number with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the lunar week (consecutive 7 day periods starting on January 1st each year) containing date.

Syntax:

```
LunarWeekName (date [, shift = 0 [, weekoffset = 0]])
```

Arguments:

Argument	Description
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the lunar week which contains date . Negative values in shift indicate preceding lunar weeks and positive values indicate succeeding lunar weeks.
weekoffset	If you want to work with an offset for the start of the lunar weeks, indicate an offset in days in of weekoffset . This may be given as a real number indicating days and/or fractions of a day.

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
<code>lunarweekname ('2006-01-12')</code>	Returns '2006/02' with an underlying numeric value corresponding to '2006-01-08 00:00:00.000'
<code>lunarweekname ('2006-01-12', -1)</code>	Returns '2006/01' with an underlying numeric value corresponding to '2006-01-01 00:00:00.000'
<code>lunarweekname ('2006-01-12', 0, 1)</code>	Returns '2006/02' with an underlying numeric value corresponding to '2006-01-09 00:00:00.000'

lunarweekstart

This function returns a value corresponding to a timestamp of the first millisecond of the lunar week (consecutive 7 day periods starting on January 1st each year) containing **date**. The default output format will be the **DateFormat** set in the script.

Syntax:

```
LunarweekStart(date [, shift = 0 [,weekoffset = 0]])
```

Arguments:

Argument	Description
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the lunar week which contains date . Negative values in shift indicate preceding lunar weeks and positive values indicate succeeding lunar weeks.
weekoffset	If you want to work with an offset for the start of the lunar weeks, indicate an offset in days in of weekoffset . This may be given as a real number indicating days and/or fractions of a day.

Examples and results:

Example	Result
<code>lunarweekstart ('2006-01-12')</code>	Returns '2006-01-08' with an underlying numeric value corresponding to '2006-01-08 00:00:00.000'
<code>lunarweekstart ('2006-01-12', -1)</code>	Returns '2006-01-01' with an underlying numeric value corresponding to '2006-01-01 00:00:00.000'
<code>lunarweekstart ('2006-01-12', 0, 1)</code>	Returns '2006-01-09' with an underlying numeric value corresponding to '2006-01-09 00:00:00.000'

makedate

This function returns a date calculated from the year **YYYY**, the month **MM** and the day **DD**.

Syntax:

```
MakeDate (YYYY [ , MM [ , DD ] ])
```

Arguments:

Argument	Description
YYYY	The year as an integer.
MM	The month as an integer. If no month is stated, 1 (January) is assumed.
DD	The day as an integer. If no day is stated, 1 (the 1:st) is assumed.

Examples and results:

Example	Result
makedate(2012)	returns 2012-01-01
makedate(12)	returns 2012-01-01
makedate(2012,12)	returns 2012-01-01
makedate(2012,2,14)	returns 2012-12-01

maketime

This function returns a time calculated from the hour **hh**, the minute **mm** the second **ss** with a fraction **fff** down to a millisecond.

Syntax:

```
MakeTime (hh [ , mm [ , ss [ .fff ] ] ])
```

Arguments:

Argument	Description
hh	The hour as an integer.
mm	The minute as an integer. If no minute is stated, 00 is assumed.
ss	The second as an integer. If no second is stated, 00 is assumed.
.fff	A fraction of a second as an integer. If no fraction of a second is stated, 000 is assumed.

5 Functions in scripts and chart expressions

Examples and results:

Example	Result
<code>maketime(22)</code>	returns 22-00-00
<code>maketime(22, 17)</code>	returns 22-17-00
<code>maketime(22, 17, 52)</code>	returns 22-17-52

makeweekdate

This function returns a date calculated from the year **YYYY**, the week **WW** and the day-of-week **D**.

Syntax:

```
MakeWeekDate (YYYY [ , WW [ , D ] ])
```

Arguments:

Argument	Description
YYYY	The year as an integer.
WW	The week as an integer.
D	The day-of-week as an integer. If no day-of-week is stated, 0 (Monday) is assumed.

Examples and results:

Example	Result
<code>makeweekdate(1999,6,6)</code>	returns 1999-02-14
<code>makeweekdate(1999,6,6)</code>	returns 1999-02-08

minute

This function returns an integer representing the minute when the fraction of the **expression** is interpreted as a time according to the standard number interpretation.

Syntax:

```
minute (expression)
```

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
<code>minute ('09:14:36')</code>	returns 14
<code>minute ('0.5555')</code>	returns 19 (Because 0.5555 = 13:19:55)

month

This function returns a dual value with a month name as defined in the environment variable **MonthNames** and an integer between 1-12. The month is calculated from the date interpretation of the expression, according to the standard number interpretation.

Syntax:

```
month(expression)
```

Examples and results:

Example	Result
<code>month('2012-10-12')</code>	returns Oct

monthend

This function returns a value corresponding to a timestamp of the last millisecond of the last day of the month containing **date**. The default output format will be the **DateFormat** set in the script.

Syntax:

```
MonthEnd(date [, shift = 0])
```

Arguments:

Argument	Description
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the month which contains date . Negative values in shift indicate preceding months and positive values indicate succeeding months.

Examples and results:

Example	Result
<code>monthend ('2001-02-19')</code>	Returns '2001-02-28' with an underlying numeric value corresponding to '2001-02-28 23:59:59.999'
<code>monthend ('2001-02-19', -1)</code>	Returns '2001-01-31' with an underlying numeric value corresponding to '2001-01-31 23:59:59.999'

monthname

This function returns a display value showing the month (formatted according to the **MonthNames** script variable) and year with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the month.

Syntax:

```
MonthName (date [, shift = 0])
```

Arguments:

Argument	Description
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the month which contains date . Negative values in shift indicate preceding months and positive values indicate succeeding months.

Examples and results:

Example	Result
monthname ('2001-10-19')	Returns 'Oct 2001' with an underlying numeric value corresponding to '2001-10-01 00:00:00.000'
monthname ('2001-10-19', -1)	Returns 'Sep 2001' with an underlying numeric value corresponding to '2001-09-01 00:00:00.000'

monthsend

This function returns a value corresponding to a timestamp of the last millisecond of the *n* month period (starting from January 1st) containing **date**. The default output format will be the **DateFormat** set in the script.

Syntax:

```
MonthsEnd (n, date [, shift = 0 [, first_month_of_year = 1]])
```

Arguments:

Argument	Description
n	n must be (1), 2, (3), 4 or 6.
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the period which contains date . Negative values in shift indicate preceding periods and positive values indicate succeeding periods.

5 Functions in scripts and chart expressions

Argument	Description
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

Examples and results:

Example	Result
monthsend (4, '2001-07-19')	Returns '2001-08-31' with an underlying numeric value corresponding to '2001-08-31 23:59:59.999'
monthsend (4, '2001-10-19', -1)	Returns '2001-08-31' with an underlying numeric value corresponding to '2001-08-31 23:59:59.999'
monthsend (4, '2001-10-19', 0, 2)	Returns '2002-01-31' with an underlying numeric value corresponding to '2002-01-31 23:59:59.999'

monthsname

This function returns a display value showing the months of the period (formatted according to the **MonthNames** script variable) and year with an underlying numeric value corresponding to a timestamp of the first millisecond of the n month period (starting from January 1st) containing **date**.

Syntax:

```
MonthsName (n, date [, shift = 0 [, first_month_of_year = 1]])
```

Arguments:

Argument	Description
n	n must be (1), 2, (3), 4 or 6.
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the period which contains date . Negative values in shift indicate preceding periods and positive values indicate succeeding periods.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

Examples and results:

Example	Result
monthsname (4, '2001-10-19')	Returns 'Sep-Dec 2001' with an underlying numeric value corresponding to '2001-09-01 00:00:00.000'

5 Functions in scripts and chart expressions

Example	Result
monthsname (4, '2001-10-19', -1)	Returns 'May-Aug 2001' with an underlying numeric value corresponding to '2001-05-01 00:00:00.000'
monthsname (4, '2001-10-19', 0, 2)	Returns 'Oct-Jan 2002' with an underlying numeric value corresponding to '2001-10-01 00:00:00.000'

monthsstart

This function returns a value corresponding to a timestamp of the first millisecond of the n month period (starting from January 1st) containing **date**. The default output format will be the **DateFormat** set in the script.

Syntax:

```
MonthsStart(n, date [, shift = 0 [, first_month_of_year = 1]])
```

Arguments:

Argument	Description
n	An integer that must be (1), 2, (3), 4 or 6.
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the period which contains date . Negative values in shift indicate preceding periods and positive values indicate succeeding periods.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

Examples and results:

Example	Result
monthsstart (4, '2001-10-19')	Returns '2001-09-01' with an underlying numeric value corresponding to '2001-09-01 00:00:00.000'
monthsstart (4, '2001-10-19', -1)	Returns '2001-05-01' with an underlying numeric value corresponding to '2001-05-01 00:00:00.000'
monthsstart (4, '2001-10-19', 0, 2)	Returns '2001-10-01' with an underlying numeric value corresponding to '2001-10-01 00:00:00.000'

monthstart

This function returns a value corresponding to a timestamp of the first millisecond of the first day of the month containing **date**. The default output format will be the **DateFormat** set in the script.

Syntax:

5 Functions in scripts and chart expressions

```
MonthStart(date [, shift = 0])
```

Arguments:

Argument	Description
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the month which contains date . Negative values in shift indicate preceding months and positive values indicate succeeding months.

Examples and results:

Example	Result
monthstart ('2001-10-19')	Returns '2001-10-01' with an underlying numeric value corresponding to '2001-10-01 00:00:00.000'
monthstart ('2001-10-19', -1)	Returns '2001-09-01' with an underlying numeric value corresponding to '2001-09-01 00:00:00.000'

networkdays

Returns the number of working days (Monday-Friday) between and including **start_date** and **end_date** taking into account any optionally listed **holiday**. All parameters should be valid dates or timestamps.

Syntax:

```
networkdays (start:date, end_date {, holiday})
```

Examples and results:

Example	Result
networkdays ('2007-02-19', '2007-03-01')	Returns 9
networkdays ('2006-12-18', '2006-12-31', '2006-12-25', '2006-12-26')	Returns 8

NOW

This function returns a timestamp of the current time from the system clock.

Syntax:

```
now([ timer_mode])
```

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
timer_mode	Can have the following values: 0 Time at script run 1 Time at function call 2 Time when the app was opened Default timer_mode is 1. The timer_mode = 1 should be used with caution, since it polls the operating system every second and hence will slow down the system.

quarterend

This function returns a value corresponding to a timestamp of the last millisecond of the quarter containing **date**. The default output format will be the **DateFormat** set in the script.

Syntax:

```
QuarterEnd(date [, shift = 0 [, first_month_of_year = 1]])
```

Arguments:

Argument	Description
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the quarter which contains date . Negative values in shift indicate preceding quarters and positive values indicate succeeding quarters.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

Examples and results:

Example	Result
quarterend ('2005-10-29')	Returns '2005-12-31' with an underlying numeric value corresponding to '2005-12-31 23:59:59.999'
quarterend('2005-10-29', -1)	Returns '2005-09-30' with an underlying numeric value corresponding to '2005-09-30 23:59:59.999'
quarterend ('2005-10-29', 0, 3)	Returns '2005-11-30' with an underlying numeric value corresponding to '2005-11-30 23:59:59.999'

quartername

This function returns a display value showing the months of the quarter (formatted according to the **MonthNames** script variable) and year with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the quarter.

5 Functions in scripts and chart expressions

Syntax:

```
QuarterName (date [, shift = 0 [, first_month_of_year = 1]])
```

Arguments:

Argument	Description
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the quarter which contains date . Negative values in shift indicate preceding quarters and positive values indicate succeeding quarters.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

Examples and results:

Example	Result
quartername ('2005-10-29')	Returns 'Oct-Dec 2005' with an underlying numeric value corresponding to '2005-10-01 00:00:00.000'
quartername ('2005-10-29', -1)	Returns 'Jul-Sep 2005' with an underlying numeric value corresponding to '2005-07-01 00:00:00.000'
quartername ('2005-10-29', 0, 3)	Returns 'Sep-Nov 2005' with an underlying numeric value corresponding to '2005-09-01 00:00:00.000'

quarterstart

This function returns a value corresponding to a timestamp of the first millisecond of the quarter containing **date**. The default output format will be the **DateFormat** set in the script.

Syntax:

```
QuarterStart (date [, shift = 0 [, first_month_of_year = 1]])
```

Arguments:

Argument	Description
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the quarter which contains date . Negative values in shift indicate preceding quarters and positive values indicate succeeding quarters.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

5 Functions in scripts and chart expressions

Examples and results:

Example	Result
quarterstart ('2005-10-29')	Returns '2005-10-01' with an underlying numeric value corresponding to '2005-10-01 00:00:00.000'
quarterstart ('2005-10-29', -1)	Returns '2005-07-01' with an underlying numeric value corresponding to '2005-07-01 00:00:00.000'
quarterstart ('2005-10-29', 0, 3)	Returns '2005-09-01' with an underlying numeric value corresponding to '2005-09-01 00:00:00.000'

second

This function returns an integer representing the second when the fraction of the **expression** is interpreted as a time according to the standard number interpretation.

Syntax:

```
second (expression)
```

Examples and results:

Example	Result
second('09:14:36')	returns 36
second('0.5555')	returns 55 (Because 0.5555 = 13:19:55)

setdateyear

This function returns a timestamp based on the input **timestamp** but with the year replaced with **year**.

Syntax:

```
setdateyear (timestamp, year)
```

Arguments:

Argument	Description
timestamp	A standard Qlik Sense timestamp (often just a date).
year	A four-digit year.

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
<code>setdateyear ('2005-10-29', 2006)</code>	Returns '2006-10-29'
<code>setdateyear ('2005-10-29 04:26', 2006)</code>	Returns '2006-10-29 04:26'

setdateyearmonth

Returns a timestamp based on the input **timestamp** but with the year replaced with **year** and the month replaced with **month**.

Syntax:

```
SetDateYearMonth (timestamp, year, month)
```

Arguments:

Argument	Description
timestamp	A standard Qlik Sense timestamp (often just a date).
year	A four-digit year.
month	A one- or two-digit month.

Examples and results:

Example	Result
<code>setdateyearmonth ('2005-10-29', 2006, 3)</code>	Returns '2006-03-29'
<code>setdateyearmonth ('2005-10-29 04:26', 2006, 3)</code>	Returns '2006-03-29 04:26'

timezone

This function returns the name of the current time zone, as defined in Windows.

Syntax:

```
TimeZone ( )
```

Example:

```
timezone( )
```

today

This function returns the current date from the system clock.

Syntax:

```
today ([ timer_mode])
```

Arguments:

Argument	Description
timer_mode	Can have the following values: 0 Time at script run 1 Time at function call 2 Time when the app was opened Default timer_mode is 1. The timer_mode = 1 should be used with caution, since it polls the operating system every second and hence will slow down the system.

UTC

Returns the current Coordinated Universal Time.

Syntax:

```
UTC ( )
```

Example:

```
utc( )
```

week

This function returns an integer representing the week number according to ISO 8601. The week number is calculated from the date interpretation of the expression, according to the standard number interpretation.

Syntax:

```
week (expression)
```

Examples and results:

Example	Result
week('2012-10-12')	returns 44

weekday

This function returns a dual value with:

- A day name as defined in the environment variable **DayNames**.
- An integer between 0-6 corresponding to the nominal day of the week (0-6).

Syntax:

5 Functions in scripts and chart expressions

```
weekday (date [, weekstart=0])
```

Arguments:

Argument	Description
date	The date to evaluate.
weekstart	<p>If you don't specify weekstart, the value of variable FirstWeekDay will be used as the first day of the week.</p> <p>If you want to use another day as the first day of the week, set weekstart to:</p> <ul style="list-style-type: none">• 0 for Monday• 1 for Tuesday• 2 for Wednesday• 3 for Thursday• 4 for Friday• 5 for Saturday• 6 for Sunday <p>The integer returned by the function will now use the first day of the week that you set with weekstart as base (0).</p> <p>See also: <i>FirstWeekDay</i> (page 110)</p>

Examples and results:

Unless it is stated specifically, **FirstWeekDay** is set to 0 in these examples.

Example	Result
<code>weekday('1971-10-12')</code>	returns 'Tue' and 1
<code>weekday('1971-10-12' , 6)</code>	returns 'Tue' and 2. In this example we use Sunday (6) as the first day of the week.
<code>SET FirstweekDay = 6;</code> ... <code>weekday('1971-10-12')</code>	returns 'Tue' and 2.

weekend

This function returns a value corresponding to a timestamp of the last millisecond of the last day (Sunday) of the calendar week containing **date**. The default output format will be the **DateFormat** set in the script.

Syntax:

```
WeekEnd (date [, shift = 0 [, weekoffset = 0]])
```


5 Functions in scripts and chart expressions

Arguments:

Argument	Description
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the week which contains date . Negative values in shift indicate preceding weeks and positive values indicate succeeding weeks.
weekoffset	If you want to work with weeks not starting midnight between Sunday and Monday, indicate an offset in days in weekoffset . This may be given as a real number indicating days and/or fractions of a day.

Examples and results:

Example	Result
weekend ('2006-01-12')	Returns '2006-01-15' with an underlying numeric value corresponding to '2006-01-15 23:59:59.999'
weekend ('2006-01-12', -1)	Returns '2006-01-08' with an underlying numeric value corresponding to '2006-01-08 23:59:59.999'
weekend ('2006-01-12', 0, 1)	Returns '2006-01-16' with an underlying numeric value corresponding to '2006-01-16 23:59:59.999'

weekname

This function returns a value showing the year and week number with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the week containing **date**.

Syntax:

```
WeekName (date [, shift = 0 [,weekoffset = 0]])
```

Arguments:

Argument	Description
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the week which contains date . Negative values in shift indicate preceding weeks and positive values indicate succeeding weeks.
weekoffset	If you want to work with weeks not starting midnight between Sunday and Monday, indicate an offset in days in weekoffset . This may be given as a real number indicating days and/or fractions of a day.

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
weekname ('2006-01-12')	Returns '2006/02' with an underlying numeric value corresponding to '2006-01-09 00:00:00.000'
weekname ('2006-01-12', -1)	Returns '2006/01' with an underlying numeric value corresponding to '2006-01-02 00:00:00.000'
weekname ('2006-01-12', 0, 1)	Returns '2006/02' with an underlying numeric value corresponding to '2006-01-10 00:00:00.000'

weekstart

This function returns a value corresponding to a timestamp of the first millisecond of the first day (Monday) of the calendar week containing **date**. The default output format is the **DateFormat** set in the script.

Syntax:

```
WeekStart(date [, shift = 0 [,weekstart = 0]])
```

Arguments:

Argument	Description
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the week which contains date . Negative values in shift indicate preceding weeks and positive values indicate succeeding weeks.
weekstart	<p>If you don't specify weekstart, the value of variable FirstWeekDay will be used as the first day of the week.</p> <p>If you want to use another day as the first day of the week, set weekstart to:</p> <ul style="list-style-type: none">• 0 for Monday• 1 for Tuesday• 2 for Wednesday• 3 for Thursday• 4 for Friday• 5 for Saturday• 6 for Sunday <p>See also: <i>FirstWeekDay</i> (page 110)</p>

Examples and results:

FirstWeekDay is set to 0 in these examples.

5 Functions in scripts and chart expressions

Example	Result
<code>weekstart ('2006-01-12')</code>	Returns '2006-01-09' with an underlying numeric value corresponding to '2006-01-09 00:00:00.000'
<code>weekstart ('2006-01-12', -1)</code>	Returns '2006-01-02' with an underlying numeric value corresponding to '2006-01-02 00:00:00.000'
<code>weekstart ('2006-01-12', 0, 1)</code>	Returns '2006-01-10' with an underlying numeric value corresponding to '2006-01-10 00:00:00.000'

weekyear

This function returns the year to which the week number belongs according to ISO 8601. The week number ranges between 1 and approximately 52.

Syntax:

```
weekyear (expression)
```

Examples and results:

Example	Result
<code>weekyear('1996-12-30')</code>	returns 1997
<code>weekyear('1997-01-02')</code>	returns 1997
<code>weekyear('1997-12-30')</code>	returns 1997
<code>weekyear('1999-01-02')</code>	returns 1998

Limitations:

Some years, week #1 starts in December, e.g. December 1997. Other years start with week #53 of previous year, e.g. January 1999. For those few days when the week number belongs to another year, the functions **year** and **weekyear** will return different values.

year

This function returns an integer representing the year when the **expression** is interpreted as a date according to the standard number interpretation.

Syntax:

```
year (expression)
```

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
<code>year('2012-10-12')</code>	returns 2012

yearend

This function returns a value corresponding to a timestamp of the last millisecond of the last day of the year containing **date**. The default output format will be the **DateFormat** set in the script.

Syntax:

```
YearEnd( date [, shift = 0 [, first_month_of_year = 1]])
```

Arguments:

Argument	Description
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the year which contains date . Negative values in shift indicate preceding years and positive values indicate succeeding years.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

Examples and results:

Example	Result
<code>yearend ('2001-10-19')</code>	Returns '2001-12-31' with an underlying numeric value corresponding to '2001-12-31 23:59:59.999'
<code>yearend ('2001-10-19', -1)</code>	Returns '2000-12-31' with an underlying numeric value corresponding to '2000-12-31 23:59:59.999'
<code>yearend ('2001-10-19', 0, 4)</code>	Returns '2002-03-31' with an underlying numeric value corresponding to '2002-03-31 23:59:59.999'

yearname

This function returns a four-digit year as display value with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the year containing **date**.

Syntax:

```
YearName(date [, shift = 0 [, first_month_of_year = 1]])
```

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the year which contains date . Negative values in shift indicate preceding years and positive values indicate succeeding years.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year . The display value will then be a string showing two years.

Examples and results:

Example	Result
<code>yearname ('2001-10-19')</code>	Returns '2001' with an underlying numeric value corresponding to '2001-01-01 00:00:00.000'
<code>yearname ('2001-10-19' , -1)</code>	Returns '2000' with an underlying numeric value corresponding to '2000-01-01 00:00:00.000'
<code>yearname ('2001-10-19' , 0, 4)</code>	Returns '2001-2002' with an underlying numeric value corresponding to '2001-04-01 00:00:00.000'

yearstart

This function returns a value corresponding to a timestamp of the first millisecond of the first day of the year containing **date**. The default output format will be the **DateFormat** set in the script.

Syntax:

```
YearStart( date [, shift = 0 [, first_month_of_year = 1]])
```

Arguments:

Argument	Description
date	The date to evaluate.
shift	shift is an integer, where the value 0 indicates the year which contains date . Negative values in shift indicate preceding years and positive values indicate succeeding years.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
<code>yearstart ('2001-10-19')</code>	Returns '2001-01-01' with an underlying numeric value corresponding to '2001-01-01 00:00:00.000'
<code>yearstart ('2001-10-19', -1)</code>	Returns '2000-01-01' with an underlying numeric value corresponding to '2000-01-01 00:00:00.000'
<code>yearstart ('2001-10-19', 0, 4)</code>	Returns '2001-04-01' with an underlying numeric value corresponding to '2001-04-01 00:00:00.000'

yeartodate

This function returns True if a **date** falls within the year to date, else False.

Syntax:

```
YearToDate (date [ , yearoffset [ , firstmonth [ , todaydate] ] ] )
```

If none of the optional parameters are used, the year to date means any date within one calendar year from January 1 up to and including the date of the last script execution.

Arguments:

Argument	Description
date	The date to evaluate as a timestamp, for example '2012-10-12'.
yearoffset	By specifying a yearoffset (0 if omitted), the function can be transposed to return True for the same period in another year. A negative yearoffset indicate previous years while a positive one indicates coming years. Last year to date is achieved by specifying yearoffset = -1.
firstmonth	By specifying a firstmonth between 1 and 12 (1 if omitted) the beginning of the year may be moved forward to the first day of any month. If you e.g. want to work with a fiscal year beginning on May 1, specify firstmonth = 5.
todaydate	By specifying a todaydate (timestamp of the last script execution if omitted) it is possible to move the day used as the upper boundary of the period.

Examples and results:

The following examples assume last reload time = 2011-11-18

Example	Result
<code>yeartodate('2010-11-18')</code>	returns False

Example	Result
<code>yeartodate('2011-02-01')</code>	returns True
<code>yeartodate('2011-11-18')</code>	returns True
<code>yeartodate('2011-11-19')</code>	returns False
<code>yeartodate('2010-11-18', -1)</code>	returns True
<code>yeartodate('2011-11-18', -1)</code>	returns False
<code>yeartodate('2011-04-30', 0, 5)</code>	returns False
<code>yeartodate('2011-05-01', 0, 5)</code>	returns True

5.6 Exponential and logarithmic functions

This section describes functions related to exponential and logarithmic calculations. All functions can be used in both the data load script and in chart expressions.

In the functions below, the parameters are expressions where x should be interpreted as a real valued number.

exp

Exponential function, with the base of the natural logarithm e as base. The result is a positive number.

```
exp (  $x$  )
```

log

The natural logarithm of x . The function is only defined if $x > 0$. The result is a number.

```
log (  $x$  )
```

log10

The 10-logarithm (base 10) of x . The function is only defined if $x > 0$. The result is a number.

```
log10 (  $x$  )
```

pow

Returns x to the power of y . The result is a number.

```
pow (  $x, y$  )
```

sqr

Square of x . The result is a number.

```
sqr (  $x$  )
```

sqrt

Square root of x . The function is only defined if $x \geq 0$. The result is a positive number.

```
sqrt(x )
```

5.7 Field functions

These functions can only be used in chart expressions.

Field functions either return integers or strings identifying different aspects of field selections.

Count functions

GetSelectedCount

GetSelectedCount() finds the number of selected (green) values in a field.

```
GetSelectedCount - chart function (field_name [, include_excluded])
```

GetAlternativeCount

GetAlternativeCount() is used to find the number of alternative (light gray) values in the identified field.

```
GetAlternativeCount - chart function (field_name)
```

GetPossibleCount

GetPossibleCount() is used to find the number of possible values in the identified field. If the identified field includes selections, then the selected (green) fields are counted. Otherwise associated (white) values are counted.

```
GetPossibleCount - chart function (field_name)
```

GetExcludedCount

GetExcludedCount() finds the number of excluded (dark gray) values in the identified field.

```
GetExcludedCount - chart function (page 395) (field_name)
```

GetNotSelectedCount

This chart function returns the number of not-selected values in the field named **fieldname**. The field must be in and-mode for this function to be relevant.

```
GetNotSelectedCount - chart function (fieldname [, includeexcluded=false])
```

Field and selection functions

GetCurrentSelections

GetCurrentSelections() returns the current selections in the app.

```
GetCurrentSelections - chart function ([record_sep [, tag_sep [, value_sep  
[, max_values]])])
```

GetFieldSelections

GetFieldSelections() returns a **string** with the current selections in a field.


```
GetFieldSelections - chart function ( field_name [, value_sep [, max_
values]])
```

GetAlternativeCount - chart function

GetAlternativeCount() is used to find the number of alternative (light gray) values in the identified field.

Syntax:

```
GetAlternativeCount (field_name)
```

Return data type: integer

Arguments:

Argument	Description
field_name	The field containing the range of data to be measured.

Examples and results:

The following example uses two fields loaded to different filter panes, one for **First name** name and one for **Initials**.

Examples	Results
Given that John is selected in First name . GetAlternativeCount ([First name])	4 as there are 4 unique and excluded (gray) values in First name .
Given that John and Peter are selected. GetAlternativeCount ([First name])	3 as there are 3 unique and excluded (gray) values in First name .
Given that no values are selected in First name . GetAlternativeCount ([First name])	0 as there are no selections.

Data used in example:

```
Initials:
LOAD * inline [
"First name"|Initials|"Has cellphone"
John|JA|Yes
Sue|SB|Yes
Mark|MC |No
Peter|PD|No
Jane|JE|Yes
Peter|PF|Yes ] (delimiter is '|');
```

GetCurrentSelections - chart function

GetCurrentSelections() returns the current selections in the app.

If options are used you will need to specify record_sep. To specify a new line set **record_sep** to **chr(13)&chr(10)**.

If all but two, or all but one, values, are selected, the format 'NOT x,y' or 'NOT y' will be used respectively. If you select all values and the count of all values is greater than max_values, the text ALL will be returned.

Syntax:

```
GetCurrentSelections ([record_sep [,tag_sep [,value_sep [,max_values]]]])
```

Return data type: string

Arguments:

Arguments	Description
record_sep	Separator to be put between field records. The default is <CR><LF> meaning a new line.
tag_sep	Separator to be put between the field name tag and the field values. The default is ': '.
value_sep	The separator to be put between field values. The default is ','.
max_values	The maximum number of field values to be individually listed. When a larger number of values is selected, the format 'x of y values' will be used instead. The default is 6.

Examples and results:

The following example uses two fields loaded to different filter panes, one for **First name** name and one for **Initials**.

Examples	Results
Given that John is selected in First name . GetCurrentSelections ()	'First name: John'
Given that John and Peter are selected in First name . GetCurrentSelections ()	'First name: John, Peter'
Given that John is selected in First name and JA is selected in Initials . GetCurrentSelections ()	'First name: John; Peter Initials: JA'
Given that John is selected in First name and JA is selected in Initials . GetCurrentSelections (chr(13)&chr(10) , ' = ')	'First name = John Initials = JA'

5 Functions in scripts and chart expressions

Examples	Results
Given that you have selected all names except Sue in First name and no selections in Initials . =GetCurrentSelections(chr(13)&chr(10),'=',',',3)	'First name=NOT Sue'

Data used in example:

```
Initials:  
LOAD * inline [  
"First name"|Initials|"Has cellphone"  
John|JA|Yes  
Sue|SB|Yes  
Mark|MC |No  
Peter|PD|No  
Jane|JE|Yes  
Peter|PF|Yes ] (delimiter is '|');
```

GetExcludedCount - chart function

GetExcludedCount() finds the number of excluded (dark gray) values in the identified field.

Syntax:

```
GetExcludedCount (field_name)
```

Return data type: string

Limitations:

GetExcludedCount() only evaluates for fields with associated values, that is, fields without selections. For fields with selections **GetExcludedCount()** will return 0.

Arguments:

Arguments	Description
field_name	The field containing the range of data to be measured.

Examples and results:

The following example uses two fields loaded to different filter panes, one for **First name** name and one for **Initials**.

Examples	Results
Given that John is selected in First name .	5 as there are 5 excluded (gray) values in Initials . The sixth cell (JA) will be white as it is associated with the selection John in First name .

5 Functions in scripts and chart expressions

Examples	Results
GetExcludedCount ([Initials])	
Given that John and Peter are selected. GetExcludedCount ([Initials])	3 as Peter is associated with 2 values in Initials .
Given that no values are selected in First name . GetExcludedCount ([Initials])	0 as there are no selections.
Given that John is selected in First name . GetExcludedCount ([First name])	0 as GetExcludedCount() only evaluates for fields with associated values, that is, fields without selections.

Data used in example:

```
Initials:
LOAD * inline [
"First name"|Initials|"Has cellphone"
John|JA|Yes
Sue|SB|Yes
Mark|MC |No
Peter|PD|No
Jane|JE|Yes
Peter|PF|Yes ] (delimiter is '|');
```

GetFieldSelections - chart function

GetFieldSelections() returns a **string** with the current selections in a field.

If all but two, or all but one of the values are selected, the format 'NOT x,y' or 'NOT y' will be used respectively. If you select all values and the count of all values is greater than max_values, the text ALL will be returned.

Syntax:

```
GetFieldSelections ( field_name [, value_sep [, max_values]])
```

Return data type: string

Arguments:

5 Functions in scripts and chart expressions

Arguments	Description
field_name	The field containing the range of data to be measured.
value_sep	The separator to be put between field values. The default is ','.
max_values	The maximum number of field values to be individually listed. When a larger number of values is selected, the format 'x of y values' will be used instead. The default is 6.

Examples and results:

The following example uses two fields loaded to different filter panes, one for **First name** name and one for **Initials**.

Examples	Results
Given that John is selected in First name . <code>GetFieldSelections ([First name])</code>	'John'
Given that John and Peter are selected. <code>GetFieldSelections ([First name])</code>	'John,Peter'
Given that John and Peter are selected. <code>GetFieldSelections ([First name],';')</code>	'John; Peter'
Given that John, Sue, Mark are selected in First name . <code>GetFieldSelections ([First name],';',2)</code>	'NOT Jane;Peter', because the value 2 is stated as the value of the max_values argument. Otherwise, the result would have been John; Sue; Mark.

Data used in example:

```
Initials:
LOAD * inline [
"First name"|Initials|"Has cellphone"
John|JA|Yes
Sue|SB|Yes
Mark|MC |No
Peter|PD|No
Jane|JE|Yes
Peter|PF|Yes ] (delimiter is '|');
```

GetNotSelectedCount - chart function

This chart function returns the number of not-selected values in the field named **fieldname**. The field must be in and-mode for this function to be relevant.

Syntax:

```
GetNotSelectedCount(fieldname [, includeexcluded=false])
```

Arguments:

Argument	Description
fieldname	The name of the field to be evaluated.
includeexcluded	If includeexcluded is stated as True, the count will include selected values which are excluded by selections in another field.

Examples:

```
GetNotSelectedCount( Country )  
GetNotSelectedCount( Country, true )
```

GetPossibleCount - chart function

GetPossibleCount() is used to find the number of possible values in the identified field. If the identified field includes selections, then the selected (green) fields are counted. Otherwise associated (white) values are counted. .

For fields with selections, **GetPossibleCount()** returns the number of selected (green) fields.

Return data type: integer

Syntax:

```
GetPossibleCount (field_name)
```

Arguments:

Arguments	Description
field_name	The field containing the range of data to be measured.

Examples and results:

The following example uses two fields loaded to different filter panes, one for **First name** name and one for **Initials**.

5 Functions in scripts and chart expressions

Examples	Results
Given that John is selected in First name . <code>GetPossibleCount ([Initials])</code>	1 as there is 1 value in Initials associated with the selection, John , in First name .
Given that John is selected in First name . <code>GetPossibleCount ([First name])</code>	1 as there is 1 selection, John , in First name .
Given that Peter is selected in First name . <code>GetPossibleCount ([Initials])</code>	2 as Peter is associated with 2 values in Initials .
Given that no values are selected in First name . <code>GetPossibleCount ([First name])</code>	5 as there are no selections and there are 5 unique values in First name .
Given that no values are selected in First name . <code>GetPossibleCount ([Initials])</code>	6 as there are no selections and there are 6 unique values in Initials .

Data used in example:

```
Initials:
LOAD * inline [
"First name"|Initials|"Has cellphone"
John|JA|Yes
Sue|SB|Yes
Mark|MC |No
Peter|PD|No
Jane|JE|Yes
Peter|PF|Yes ] (delimiter is '|');
```

GetSelectedCount - chart function

GetSelectedCount() finds the number of selected (green) values in a field.

Syntax:

```
GetSelectedCount (field_name [, include_excluded])
```

Return data type: integer

Arguments:

5 Functions in scripts and chart expressions

Arguments	Description
field_name	The field containing the range of data to be measured.
include_excluded	If set to True() , the count will include selected values, which are currently excluded by selections in other fields. If False or omitted, these values will not be included

Examples and results:

The following example uses three fields loaded to different filter panes, one for **First name** name, one for **Initials** and one for **Has cellphone**.

Examples	Results
Given that John is selected in First name . <code>getSelectedCount ([First name])</code>	1 as one value is selected in First name .
Given that John is selected in First name . <code>getSelectedCount ([Initials])</code>	0 as no values are selected in Initials .
With no selections in First name , select all values in Initials and after that select the value Yes in Has cellphone . <code>getSelectedCount ([Initials])</code>	6. Although selections with Initials MC and PD have Has cellphone set to No , the result is still 6, because the argument <code>include_excluded</code> is set to <code>True()</code> .

Data used in example:

```
Initials:
LOAD * inline [
"First name"|Initials|"Has cellphone"
John|JA|Yes
Sue|SB|Yes
Mark|MC |No
Peter|PD|No
Jane|JE|Yes
Peter|PF|Yes ] (delimiter is '|');
```

5.8 File functions

The file functions (only available in script expressions) return information about the table file which is currently being read. These functions will return NULL for all data sources except table files (exception: **ConnectString()**).

File functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Attribute

5 Functions in scripts and chart expressions

This script function returns the value of the meta tags of different media files as text. The following file formats are supported: MP3, WMA, WMV, PNG and JPG. If the file **filename** does not exist, is not a supported file format or does not contain a meta tag named **attributename**, NULL will be returned.

```
Attribute (filename, attributename)
```

ConnectString

This script function returns the active **connect** string for ODBC or OLE DB connections. Returns an empty string if no **connect** statement has been executed or after a **disconnect** statement.

```
ConnectString ()
```

FileBaseName

This script function returns a string containing the name of the table file currently being read, without path or extension.

```
FileBaseName ()
```

FileDir

This script function returns a string containing the path to the directory of the table file currently being read.

```
FileDir ()
```

FileExtension

This script function returns a string containing the extension of the table file currently being read.

```
FileExtension ()
```

FileName

This script function returns a string containing the name of the table file currently being read, without path but including the extension.

```
FileName ()
```

FilePath

This script function returns a string containing the full path to the table file currently being read.

```
FilePath ()
```

FileSize

This script function returns an integer containing the size in bytes of the file **filename** or, if no **filename** is specified, of the table file currently being read.

```
FileSize ()
```

FileTime

This script function returns a timestamp for the date and time of the last modification of the file **filename**. If no **filename** is specified, the function will refer to the currently read table file.

```
FileTime ([ filename ])
```

5 Functions in scripts and chart expressions

GetFolderPath

This script function returns the value of the Microsoft WindowsSHGetFolderPath function and returns the path. For example, **MyMusic**. Note that the function does not use the spaces seen in Windows Explorer.

```
GetFolderPath ()
```

QvdCreateTime

This script function returns the XML-header time stamp from a QVD file if any (otherwise NULL).

```
QvdCreateTime (filename)
```

QvdFieldName

This script function returns the name of field number **fieldno**, if it exists in a QVD file (otherwise NULL).

```
QvdFieldName (filename , fieldno)
```

QvdNoOfFields

This script function returns the number of fields in a QVD file.

```
QvdNoOfFields (filename)
```

QvdNoOfRecords

This script function returns the number of records currently in a QVD file.

```
QvdNoOfRecords (filename)
```

QvdTableName

This script function returns the name of the table contained in a QVD file.

```
QvdTableName (filename)
```

Attribute

This script function returns the value of the meta tags of different media files as text. The following file formats are supported: MP3, WMA, WMV, PNG and JPG. If the file **filename** does not exist, is not a supported file format or does not contain a meta tag named **attributename**, NULL will be returned.

Syntax:

```
Attribute(filename, attributename)
```

A large number of meta tags can be used, for example 'Artist' or 'Date Picture Taken'.



You can only read meta tags saved in the file according to the relevant specification, for example ID2v3 for MP3 files or EXIF for JPG files, not meta information saved in the Windows File Explorer.

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
filename	<p>The name of a media file including path, if needed, as a folder data connection.</p> <p>Example: <i>'lib://Table Files/'</i></p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none"> absolute <p>Example: <i>c:\data</i></p> <ul style="list-style-type: none"> relative to the Qlik Sense app working directory. <p>Example: <i>data</i></p>
attributename	The name of a meta tag.

Examples and results:

Example	Result
<pre>// Script to read MP3 meta tags for each vExt in 'mp3' for each vFoundFile in filelist(GetFolderPath('MyMusic') & '*.' & vExt) FileList: LOAD FileLongName, subfield(FileLongName, '\', -1) as FileShortName, num(FileSize(FileLongName), '# ### ## #' , ', ') as FileSize, FileTime(FileLongName) as FileTime, // ID3v1.0 and ID3v1.1 tags Attribute(FileLongName, 'Title') as Title, Attribute(FileLongName, 'Artist') as Artist, Attribute(FileLongName, 'Album') as Album, Attribute(FileLongName, 'Year') as Year, Attribute(FileLongName, 'Comment') as Comment, Attribute(FileLongName, 'Track') as Track, Attribute(FileLongName, 'Genre') as Genre, // ID3v2.3 tags Attribute(FileLongName, 'AENC') as AENC, // Audio encryption Attribute(FileLongName, 'APIC') as APIC, // Attached picture Attribute(FileLongName, 'COMM') as COMM, // Comments Attribute(FileLongName, 'COMR') as COMR, // Commercial frame Attribute(FileLongName, 'ENCR') as ENCR, // Encryption method registration Attribute(FileLongName, 'EQUA') as EQUA, // Equalization Attribute(FileLongName, 'ETCO') as ETCO, // Event timing codes Attribute(FileLongName, 'GEOB') as GEOB, // General encapsulated object Attribute(FileLongName, 'GRID') as GRID, // Group identification registration</pre>	<p>Script to read all possible MP3 meta tags in folder <i>MyMusic</i></p>

5 Functions in scripts and chart expressions

Example	Result
<pre> Attribute(FileLongName, 'IPLS') as IPLS, // Involved people list Attribute(FileLongName, 'LINK') as LINK, // Linked information Attribute(FileLongName, 'MCDI') as MCDI, // Music CD identifier Attribute(FileLongName, 'MLLT') as MLLT, // MPEG location lookup table Attribute(FileLongName, 'OWNE') as OWNE, // Ownership frame Attribute(FileLongName, 'PRIV') as PRIV, // Private frame Attribute(FileLongName, 'PCNT') as PCNT, // Play counter Attribute(FileLongName, 'POPM') as POPM, // Popularimeter Attribute(FileLongName, 'POSS') as POSS, // Position synchronisation frame Attribute(FileLongName, 'RBUF') as RBUF, // Recommended buffer size Attribute(FileLongName, 'RVAD') as RVAD, // Relative volume adjustment Attribute(FileLongName, 'RVRB') as RVRB, // Reverb Attribute(FileLongName, 'SYLT') as SYLT, // Synchronized lyric/text Attribute(FileLongName, 'SYTC') as SYTC, // Synchronized tempo codes Attribute(FileLongName, 'TALB') as TALB, // Album/Movie/Show title Attribute(FileLongName, 'TBPM') as TBPM, // BPM (beats per minute) Attribute(FileLongName, 'TCOM') as TCOM, // Composer Attribute(FileLongName, 'TCON') as TCON, // Content type Attribute(FileLongName, 'TCOP') as TCOP, // Copyright message Attribute(FileLongName, 'TDAT') as TDAT, // Date Attribute(FileLongName, 'TDLY') as TDLY, // Playlist delay Attribute(FileLongName, 'TENC') as TENC, // Encoded by Attribute(FileLongName, 'TEXT') as TEXT, // Lyricist/Text writer Attribute(FileLongName, 'TFLT') as TFLT, // File type Attribute(FileLongName, 'TIME') as TIME, // Time Attribute(FileLongName, 'TIT1') as TIT1, // Content group description Attribute(FileLongName, 'TIT2') as TIT2, // Title/songname/content description Attribute(FileLongName, 'TIT3') as TIT3, // Subtitle/Description refinement Attribute(FileLongName, 'TKEY') as TKEY, // Initial key Attribute(FileLongName, 'TLAN') as TLAN, // Language(s) Attribute(FileLongName, 'TLEN') as TLEN, // Length Attribute(FileLongName, 'TMED') as TMED, // Media type Attribute(FileLongName, 'TOAL') as TOAL, // Original album/movie/show title Attribute(FileLongName, 'TOFN') as TOFN, // Original filename Attribute(FileLongName, 'TOLY') as TOLY, // Original lyricist(s)/text writer(s) Attribute(FileLongName, 'TOPE') as TOPE, // Original artist(s)/performer(s) Attribute(FileLongName, 'TORY') as TORY, // Original release year Attribute(FileLongName, 'TOWN') as TOWN, // File owner/licensee Attribute(FileLongName, 'TPE1') as TPE1, // Lead performer(s)/Soloist(s) Attribute(FileLongName, 'TPE2') as TPE2, // Band/orchestra/accompaniment Attribute(FileLongName, 'TPE3') as TPE3, // Conductor/performer refinement Attribute(FileLongName, 'TPE4') as TPE4, // Interpreted, remixed, or otherwise modified by Attribute(FileLongName, 'TPOS') as TPOS, // Part of a set Attribute(FileLongName, 'TPUB') as TPUB, // Publisher Attribute(FileLongName, 'TRCK') as TRCK, // Track number/Position in set Attribute(FileLongName, 'TRDA') as TRDA, // Recording dates Attribute(FileLongName, 'TRSN') as TRSN, // Internet radio station name Attribute(FileLongName, 'TRSO') as TRSO, // Internet radio station owner Attribute(FileLongName, 'TSIZ') as TSIZ, // Size </pre>	

5 Functions in scripts and chart expressions

Example	Result
<pre>// Script to read Jpeg Exif meta tags for each vExt in 'jpg', 'jpeg', 'jpe', 'jfif', 'jif', 'jfi' for each vFoundFile in fileList(GetFolderPath('MyPictures') & '*.' & vExt) FileList: LOAD FileLongName, subfield(FileLongName, '\', -1) as FileShortName, num(FileSize(FileLongName), '# ### ##', ',', ' ') as FileSize, FileTime(FileLongName) as FileTime, // ***** Exif Main (IFD0) Attributes ***** Attribute(FileLongName, 'ImageWidth') as ImageWidth, Attribute(FileLongName, 'ImageLength') as ImageLength, Attribute(FileLongName, 'BitsPerSample') as BitsPerSample, Attribute(FileLongName, 'Compression') as Compression, // examples: 1=uncompressed, 2=CCITT, 3=CCITT 3, 4=CCITT 4, // 5=LZW, 6=JPEG (old style), 7=JPEG, 8=Deflate, 32773=PackBits RLE, Attribute(FileLongName, 'PhotometricInterpretation') as PhotometricInterpretation, // examples: 0=WhiteIsZero, 1=BlackIsZero, 2=RGB, 3=Palette, 5=CMYK, 6=YCbCr, Attribute(FileLongName, 'ImageDescription') as ImageDescription, Attribute(FileLongName, 'Make') as Make, Attribute(FileLongName, 'Model') as Model, Attribute(FileLongName, 'StripOffsets') as StripOffsets, Attribute(FileLongName, 'Orientation') as Orientation, // examples: 1=TopLeft, 2=TopRight, 3=BottomRight, 4=BottomLeft, // 5=LeftTop, 6=RightTop, 7=RightBottom, 8=LeftBottom, Attribute(FileLongName, 'SamplesPerPixel') as SamplesPerPixel, Attribute(FileLongName, 'RowsPerStrip') as RowsPerStrip, Attribute(FileLongName, 'StripByteCounts') as StripByteCounts, Attribute(FileLongName, 'XResolution') as XResolution, Attribute(FileLongName, 'YResolution') as YResolution, Attribute(FileLongName, 'PlanarConfiguration') as PlanarConfiguration, // examples: 1=chunky format, 2=planar format, Attribute(FileLongName, 'ResolutionUnit') as ResolutionUnit, // examples: 1=none, 2=inches, 3=centimeters, Attribute(FileLongName, 'TransferFunction') as TransferFunction, Attribute(FileLongName, 'Software') as Software, Attribute(FileLongName, 'DateTime') as DateTime, Attribute(FileLongName, 'Artist') as Artist, Attribute(FileLongName, 'HostComputer') as HostComputer, Attribute(FileLongName, 'WhitePoint') as WhitePoint, Attribute(FileLongName, 'PrimaryChromaticities') as PrimaryChromaticities, Attribute(FileLongName, 'YCbCrCoefficients') as YCbCrCoefficients, Attribute(FileLongName, 'YCbCrSubSampling') as YCbCrSubSampling, Attribute(FileLongName, 'YCbCrPositioning') as YCbCrPositioning, // examples: 1=centered, 2=co-sited, Attribute(FileLongName, 'ReferenceBlackWhite') as ReferenceBlackWhite, Attribute(FileLongName, 'Rating') as Rating, Attribute(FileLongName, 'RatingPercent') as RatingPercent, Attribute(FileLongName, 'ThumbnailFormat') as ThumbnailFormat, // examples: 0=Raw Rgb, 1=Jpeg, Attribute(FileLongName, 'Copyright') as Copyright,</pre>	<p>Script to read all possible EXIF meta tags from JPG files in folder <i>MyMusic</i></p>

5 Functions in scripts and chart expressions

Example	Result
<pre> Attribute(FileLongName, 'ExposureTime') as ExposureTime, Attribute(FileLongName, 'FNumber') as FNumber, Attribute(FileLongName, 'ExposureProgram') as ExposureProgram, // examples: 0=Not defined, 1=Manual, 2=Normal program, 3=Aperture priority, 4=Shutter priority, // 5=Creative program, 6=Action program, 7=Portrait mode, 8=Landscape mode, 9=Bulb, Attribute(FileLongName, 'ISOSpeedRatings') as ISOSpeedRatings, Attribute(FileLongName, 'TimeZoneOffset') as TimeZoneOffset, Attribute(FileLongName, 'SensitivityType') as SensitivityType, // examples: 0=Unknown, 1=Standard output sensitivity (SOS), 2=Recommended exposure index (REI), // 3=ISO speed, 4=Standard output sensitivity (SOS) and Recommended exposure index (REI), //5=Standard output sensitivity (SOS) and ISO Speed, 6=Recommended exposure index (REI) and ISO Speed, // 7=Standard output sensitivity (SOS) and Recommended exposure index (REI) and ISO speed, Attribute(FileLongName, 'ExifVersion') as ExifVersion, Attribute(FileLongName, 'DateTimeOriginal') as DateTimeOriginal, Attribute(FileLongName, 'DateTimeDigitized') as DateTimeDigitized, Attribute(FileLongName, 'ComponentsConfiguration') as ComponentsConfiguration, // examples: 1=Y, 2=Cb, 3=Cr, 4=R, 5=G, 6=B, Attribute(FileLongName, 'CompressedBitsPerPixel') as CompressedBitsPerPixel, Attribute(FileLongName, 'ShutterSpeedValue') as ShutterSpeedValue, Attribute(FileLongName, 'ApertureValue') as ApertureValue, Attribute(FileLongName, 'BrightnessValue') as BrightnessValue, // examples: - 1=Unknown, Attribute(FileLongName, 'ExposureBiasValue') as ExposureBiasValue, Attribute(FileLongName, 'MaxApertureValue') as MaxApertureValue, Attribute(FileLongName, 'SubjectDistance') as SubjectDistance, // examples: 0=Unknown, -1=Infinity, Attribute(FileLongName, 'MeteringMode') as MeteringMode, // examples: 0=Unknown, 1=Average, 2=CenterWeightedAverage, 3=Spot, // 4=MultiSpot, 5=Pattern, 6=Partial, 255=Other, Attribute(FileLongName, 'LightSource') as LightSource, // examples: 0=Unknown, 1=Daylight, 2=Fluorescent, 3=Tungsten, 4=Flash, 9=Fine weather, // 10=Cloudy weather, 11=Shade, 12=Daylight fluorescent, // 13=Day white fluorescent, 14=Cool white fluorescent, // 15=white fluorescent, 17=Standard light A, 18=Standard light B, 19=Standard light C, // 20=D55, 21=D65, 22=D75, 23=D50, 24=ISO studio tungsten, 255=other light source, Attribute(FileLongName, 'Flash') as Flash, Attribute(FileLongName, 'FocalLength') as FocalLength, Attribute(FileLongName, 'SubjectArea') as SubjectArea, Attribute(FileLongName, 'MakerNote') as MakerNote, Attribute(FileLongName, 'UserComment') as UserComment, Attribute(FileLongName, 'SubSecTime') as SubSecTime, Attribute(FileLongName, 'SubSecTimeOriginal') as SubSecTimeOriginal, Attribute(FileLongName, 'SubSecTimeDigitized') as SubSecTimeDigitized, Attribute(FileLongName, 'XPTitle') as XPTitle, </pre>	

5 Functions in scripts and chart expressions

Example	Result
<pre> Attribute(FileLongName, 'XPComment') as XPComment, Attribute(FileLongName, 'XPAuthor') as XPAuthor, Attribute(FileLongName, 'XPKeywords') as XPKeywords, Attribute(FileLongName, 'XPSubject') as XPSubject, Attribute(FileLongName, 'FlashpixVersion') as FlashpixVersion, Attribute(FileLongName, 'ColorSpace') as ColorSpace, // examples: 1=sRGB, 65535=Uncalibrated, Attribute(FileLongName, 'PixelXDimension') as PixelXDimension, Attribute(FileLongName, 'PixelYDimension') as PixelYDimension, Attribute(FileLongName, 'RelatedSoundFile') as RelatedSoundFile, Attribute(FileLongName, 'FocalPlaneXResolution') as FocalPlaneXResolution, Attribute(FileLongName, 'FocalPlaneYResolution') as FocalPlaneYResolution, Attribute(FileLongName, 'FocalPlaneResolutionUnit') as FocalPlaneResolutionUnit, // examples: 1=None, 2=Inch, 3=Centimeter, Attribute(FileLongName, 'ExposureIndex') as ExposureIndex, Attribute(FileLongName, 'SensingMethod') as SensingMethod, // examples: 1=Not defined, 2=One-chip color area sensor, 3=Two-chip color area sensor, // 4=Three-chip color area sensor, 5=Color sequential area sensor, // 7=Trilinear sensor, 8=Color sequential linear sensor, Attribute(FileLongName, 'FileSource') as FileSource, // examples: 0=Other, 1=Scanner of transparent type, // 2=Scanner of reflex type, 3=Digital still camera, Attribute(FileLongName, 'SceneType') as SceneType, // examples: 1=A directly photographed image, Attribute(FileLongName, 'CFAPattern') as CFAPattern, Attribute(FileLongName, 'CustomRendered') as CustomRendered, // examples: 0=Normal process, 1=Custom process, Attribute(FileLongName, 'ExposureMode') as ExposureMode, // examples: 0=Auto exposure, 1=Manual exposure, 2=Auto bracket, Attribute(FileLongName, 'WhiteBalance') as WhiteBalance, // examples: 0=Auto white balance, 1=Manual white balance, Attribute(FileLongName, 'DigitalZoomRatio') as DigitalZoomRatio, Attribute(FileLongName, 'FocalLengthIn35mmFilm') as FocalLengthIn35mmFilm, Attribute(FileLongName, 'SceneCaptureType') as SceneCaptureType, // examples: 0=Standard, 1=Landscape, 2=Portrait, 3=Night scene, Attribute(FileLongName, 'GainControl') as GainControl, // examples: 0=None, 1=Low gain up, 2=High gain up, 3=Low gain down, 4=High gain down, Attribute(FileLongName, 'Contrast') as Contrast, // examples: 0=Normal, 1=Soft, 2=Hard, Attribute(FileLongName, 'Saturation') as Saturation, // examples: 0=Normal, 1=Low saturation, 2=High saturation, Attribute(FileLongName, 'Sharpness') as Sharpness, // examples: 0=Normal, 1=Soft, 2=Hard, Attribute(FileLongName, 'SubjectDistanceRange') as SubjectDistanceRange, // examples: 0=Unknown, 1=Macro, 2=Close view, 3=Distant view, Attribute(FileLongName, 'ImageUniqueID') as ImageUniqueID, Attribute(FileLongName, 'BodySerialNumber') as BodySerialNumber, Attribute(FileLongName, 'CMNT_GAMMA') as CMNT_GAMMA, Attribute(FileLongName, 'PrintImageMatching') as PrintImageMatching, </pre>	

5 Functions in scripts and chart expressions

Example	Result
<pre> Attribute(FileLongName, 'OffsetSchema') as OffsetsSchema, // ***** Interoperability Attributes ***** Attribute(FileLongName, 'InteroperabilityIndex') as InteroperabilityIndex, Attribute(FileLongName, 'InteroperabilityVersion') as InteroperabilityVersion, Attribute(FileLongName, 'InteroperabilityRelatedImageFileFormat') as InteroperabilityRelatedImageFileFormat, Attribute(FileLongName, 'InteroperabilityRelatedImagewidth') as InteroperabilityRelatedImagewidth, Attribute(FileLongName, 'InteroperabilityRelatedImageLength') as InteroperabilityRelatedImageLength, Attribute(FileLongName, 'InteroperabilityColorSpace') as InteroperabilityColorSpace, // examples: 1=sRGB, 65535=Uncalibrated, Attribute(FileLongName, 'InteroperabilityPrintImageMatching') as InteroperabilityPrintImageMatching, // ***** GPS Attributes ***** Attribute(FileLongName, 'GPSVersionID') as GPSVersionID, Attribute(FileLongName, 'GPSLatitudeRef') as GPSLatitudeRef, Attribute(FileLongName, 'GPSLatitude') as GPSLatitude, Attribute(FileLongName, 'GPSLongitudeRef') as GPSLongitudeRef, Attribute(FileLongName, 'GPSLongitude') as GPSLongitude, Attribute(FileLongName, 'GPSAltitudeRef') as GPSAltitudeRef, // examples: 0=Above sea level, 1=Below sea level, Attribute(FileLongName, 'GPSAltitude') as GPSAltitude, Attribute(FileLongName, 'GPSTimeStamp') as GPSTimeStamp, Attribute(FileLongName, 'GPSSatellites') as GPSSatellites, Attribute(FileLongName, 'GPSStatus') as GPSStatus, Attribute(FileLongName, 'GPSMeasureMode') as GPSMeasureMode, Attribute(FileLongName, 'GPSDOP') as GPSDOP, Attribute(FileLongName, 'GPSSpeedRef') as GPSSpeedRef, Attribute(FileLongName, 'GPSSpeed') as GPSSpeed, Attribute(FileLongName, 'GPSTrackRef') as GPSTrackRef, Attribute(FileLongName, 'GPSTrack') as GPSTrack, Attribute(FileLongName, 'GPSImgDirectionRef') as GPSImgDirectionRef, Attribute(FileLongName, 'GPSImgDirection') as GPSImgDirection, Attribute(FileLongName, 'GPSMapDatum') as GPSMapDatum, Attribute(FileLongName, 'GPSDestLatitudeRef') as GPSDestLatitudeRef, Attribute(FileLongName, 'GPSDestLatitude') as GPSDestLatitude, Attribute(FileLongName, 'GPSDestLongitudeRef') as GPSDestLongitudeRef, Attribute(FileLongName, 'GPSDestLongitude') as GPSDestLongitude, Attribute(FileLongName, 'GPSDestBearingRef') as GPSDestBearingRef, Attribute(FileLongName, 'GPSDestBearing') as GPSDestBearing, Attribute(FileLongName, 'GPSDestDistanceRef') as GPSDestDistanceRef, Attribute(FileLongName, 'GPSDestDistance') as GPSDestDistance, Attribute(FileLongName, 'GPSProcessingMethod') as GPSProcessingMethod, Attribute(FileLongName, 'GPSAreaInformation') as GPSAreaInformation, Attribute(FileLongName, 'GPSDateStamp') as GPSDateStamp, Attribute(FileLongName, 'GPSDifferential') as GPSDifferential; // examples: 0=No correction, 1=Differential correction, LOAD @1:n as FileLongName Inline "\$(vFoundFile)" (fix, no labels); Next vFoundFile </pre>	

5 Functions in scripts and chart expressions

Example	Result
<pre> Next vExt / Script to read WMA/WMV ASF meta tags for each vExt in 'asf', 'wma', 'wmv' for each vFoundFile in fileList(GetFolderPath('MyMusic') & '*.' & vExt) FileList: LOAD FileLongName, subfield(FileLongName, '\', -1) as FileShortName, num(FileSize(FileLongName), '# ### ##', ',', ' ') as FileSize, FileTime(FileLongName) as FileTime, Attribute(FileLongName, 'Title') as Title, Attribute(FileLongName, 'Author') as Author, Attribute(FileLongName, 'Copyright') as Copyright, Attribute(FileLongName, 'Description') as Description, Attribute(FileLongName, 'Rating') as Rating, Attribute(FileLongName, 'PlayDuration') as PlayDuration, Attribute(FileLongName, 'MaximumBitrate') as MaximumBitrate, Attribute(FileLongName, 'WMFSDKVersion') as WMFSDKVersion, Attribute(FileLongName, 'WMFSDKNeeded') as WMFSDKNeeded, Attribute(FileLongName, 'IsVBR') as IsVBR, Attribute(FileLongName, 'ASFLeakyBucketPairs') as ASFLeakyBucketPairs, Attribute(FileLongName, 'PeakValue') as PeakValue, Attribute(FileLongName, 'AverageLevel') as AverageLevel; LOAD @1:n as FileLongName Inline "\$(vFoundFile)" (fix, no labels); Next vFoundFile Next vExt </pre>	<p>Script to read all possible WMA/WMV ASF meta tags in folder <i>MyMusic</i></p>
<pre> // Script to read PNG meta tags for each vExt in 'png' for each vFoundFile in fileList(GetFolderPath('MyPictures') & '*.' & vExt) FileList: LOAD FileLongName, subfield(FileLongName, '\', -1) as FileShortName, num(FileSize(FileLongName), '# ### ##', ',', ' ') as FileSize, FileTime(FileLongName) as FileTime, Attribute(FileLongName, 'Comment') as Comment, Attribute(FileLongName, 'Creation Time') as Creation_Time, Attribute(FileLongName, 'Source') as Source, Attribute(FileLongName, 'Title') as Title, Attribute(FileLongName, 'Software') as Software, Attribute(FileLongName, 'Author') as Author, Attribute(FileLongName, 'Description') as Description, Attribute(FileLongName, 'Copyright') as Copyright; LOAD @1:n as FileLongName Inline "\$(vFoundFile)" (fix, no labels); Next vFoundFile Next vExt </pre>	<p>Script to read all possible PNG meta tags in folder <i>MyPictures</i></p>

ConnectString

This script function returns the active **connect** string for ODBC or OLE DB connections. Returns an empty string if no **connect** statement has been executed or after a **disconnect** statement.

Syntax:

ConnectionString ()



This function supports only folder data connections in standard mode.

FileName

This script function returns a string containing the name of the table file currently being read, without path or extension.

Syntax:

FileName ()

Examples and results:

Example	Result
LOAD *, filename() as X from C:\UserFiles\abc.txt	Will return 'abc' in field X in each record read.

FileDir

This script function returns a string containing the path to the directory of the table file currently being read.

Syntax:

FileDir ()



This function supports only folder data connections in standard mode.

Examples and results:

Example	Result
Load *, filedir() as X from C:\UserFiles\abc.txt	Will return 'C:\UserFiles' in field X in each record read.

FileExtension

This script function returns a string containing the extension of the table file currently being read.

Syntax:

FileExtension ()

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
LOAD *, FileExtension() as X from C:\UserFiles\abc.txt	Will return 'txt' in field X in each record read.

FileName

This script function returns a string containing the name of the table file currently being read, without path but including the extension.

Syntax:

```
FileName ( )
```

Examples and results:

Example	Result
LOAD *, FileName() as X from C:\UserFiles\abc.txt	Will return 'abc.txt' in field X in each record read.

FilePath

This script function returns a string containing the full path to the table file currently being read.

Syntax:

```
FilePath ( )
```



This function supports only folder data connections in standard mode.

Examples and results:

Example	Result
Load *, FilePath() as X from C:\UserFiles\abc.txt	Will return 'C:\UserFiles\abc.txt' in field X in each record read.

FileSize

This script function returns an integer containing the size in bytes of the file **filename** or, if no **filename** is specified, of the table file currently being read.

Syntax:

```
FileSize ([filename])
```

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
filename	<p>The name of a file, if necessary including path, as a folder or web file data connection.</p> <p>Example: 'lib://Table Files'</p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none">absolute <p>Example: c:\data\</p> <ul style="list-style-type: none">relative to the Qlik Sense app working directory. <p>Example: data\</p> <ul style="list-style-type: none">URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. <p>Example: http://www.qlik.com</p>

Examples and results:

Example	Result
<code>LOAD *, FileSize() as X from abc.txt;</code>	Will return the size of the specified file (abc.txt) as an integer in field X in each record read.
<code>FileSize('lib://MyData/xyz.xls')</code>	Will return the size of the file xyz.xls.

FileTime

This script function returns a timestamp for the date and time of the last modification of the file **filename**. If no **filename** is specified, the function will refer to the currently read table file.

Syntax:

```
FileTime( [ filename ] )
```

Arguments:

Argument	Description
filename	<p>The name of a file, if necessary including path, as a folder or web file data connection.</p> <p>Example: 'lib://Table Files'</p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none">absolute

5 Functions in scripts and chart expressions

Argument	Description
	<p>Example: c:\data\</p> <ul style="list-style-type: none">relative to the Qlik Sense app working directory. <p>Example: data\</p> <ul style="list-style-type: none">URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. <p>Example: http://www.qlik.com</p>

Examples and results:

Example	Result
<code>LOAD *, FileTime() as X from abc.txt;</code>	Will return the date and time of the last modification of the file (abc.txt) as a timestamp in field X in each record read.
<code>FileTime('xyz.xls')</code>	Will return the timestamp of the last modification of the file xyz.xls.

GetFolderPath

This script function returns the value of the Microsoft WindowsSHGetFolderPath function and returns the path. For example, **MyMusic**. Note that the function does not use the spaces seen in Windows Explorer.



This function is not supported in standard mode.

Syntax:

```
GetFolderPath ( )
```

Examples:

```
GetFolderPath('MyMusic')  
GetFolderPath('MyPictures')  
GetFolderPath('MyVideos')  
GetFolderPath('MyReceivedFiles')  
GetFolderPath('MyShapes')  
GetFolderPath('ProgramFiles')  
GetFolderPath('windows')
```

QvdCreateTime

This script function returns the XML-header time stamp from a QVD file if any (otherwise NULL).

Syntax:

```
QvdCreateTime (filename)
```

Arguments:

Argument	Description
filename	<p>The name of a QVD file, if necessary including path, as a folder or web data connection.</p> <p>Example: 'lib://Table Files/'</p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none">• absolute <p>Example: c:\data\</p> <ul style="list-style-type: none">• relative to the Qlik Sense app working directory. <p>Example: data\</p> <ul style="list-style-type: none">• URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. <p>Example: http://www.qlik.com</p>

Example:

```
QvdCreateTime('MyFile.qvd')
QvdCreateTime('C:\MyDir\MyFile.qvd')
```

QvdFieldName

This script function returns the name of field number **fieldno**, if it exists in a QVD file (otherwise NULL).

Syntax:

```
QvdFieldName (filename , fieldno)
```

Arguments:

Argument	Description
filename	<p>The name of a QVD file, if necessary including path, as a folder or web data connection.</p> <p>Example: 'lib://Table Files/'</p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none">• absolute <p>Example: c:\data\</p>

5 Functions in scripts and chart expressions

Argument	Description
	<ul style="list-style-type: none">relative to the Qlik Sense app working directory. Example: <i>data</i>URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. Example: <i>http://www.qlik.com</i>
fieldno	The number of the field (0 based) within the table contained in the QVD file.

Examples:

```
QvdFieldName ('MyFile.qvd', 3)
QvdFieldName ('C:\MyDir\MyFile.qvd', 5)
```

QvdNoOfFields

This script function returns the number of fields in a QVD file.

Syntax:

```
QvdNoOfFields (filename)
```

Arguments:

Argument	Description
filename	<p>The name of a QVD file, if necessary including path, as a folder or web data connection.</p> <p>Example: <i>'lib://Table Files/'</i></p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none">absolute Example: <i>c:\data</i>relative to the Qlik Sense app working directory. Example: <i>data</i>URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. Example: <i>http://www.qlik.com</i>

Examples:

```
QvdNoOfFields ('MyFile.qvd')
QvdNoOfFields ('C:\MyDir\MyFile.qvd')
```

QvdNoOfRecords

This script function returns the number of records currently in a QVD file.

Syntax:

```
QvdNoOfRecords (filename)
```

Arguments:

Argument	Description
filename	<p>The name of a QVD file, if necessary including path, as a folder or web data connection.</p> <p>Example: 'lib://Table Files/'</p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none">absolute <p>Example: c:\data\</p> <ul style="list-style-type: none">relative to the Qlik Sense app working directory. <p>Example: data\</p> <ul style="list-style-type: none">URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. <p>Example: http://www.qlik.com</p>

Examples:

```
QvdNoOfRecords ('MyFile.qvd')
```

```
QvdNoOfRecords ('C:\MyDir\MyFile.qvd')
```

QvdTableName

This script function returns the name of the table contained in a QVD file.

Syntax:

```
QvdTableName (filename)
```

Arguments:

Argument	Description
filename	The name of a QVD file, if necessary including path, as a folder or web data connection.

5 Functions in scripts and chart expressions

Argument	Description
	<p>Example: 'lib://Table Files/'</p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none">absolute <p>Example: c:\data</p> <ul style="list-style-type: none">relative to the Qlik Sense app working directory. <p>Example: data</p> <ul style="list-style-type: none">URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. <p>Example: http://www.qlik.com</p>

Examples:

```
QvdTableName ('MyFile.qvd')
```

```
QvdTableName ('C:\MyDir\MyFile.qvd')
```

5.9 Financial functions

Financial functions can be used in the data load script and in chart expressions to calculate payments and interest rates.

For all the arguments, cash that is paid out is represented by negative numbers. Cash received is represented by positive numbers.

Listed here are the arguments that are used in the financial functions (excepting the ones beginning with **range-**).



*For all financial functions it is vital that you are consistent when specifying units for **rate** and **nper**. If monthly payments are made on a five-year loan at 6% annual interest, use 0.005 (6%/12) for **rate** and 60 (5*12) for **nper**. If annual payments are made on the same loan, use 6% for **rate** and 5 for **nper**.*

Financial functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

FV

This function returns the future value of an investment based on periodic, constant payments and a constant interest rate. The result has a default number format of **money**.

5 Functions in scripts and chart expressions

```
FV (rate, nper, pmt [ ,pv [ , type ] ])
```

nPer

This function returns the number of periods for an investment based on periodic, constant payments and a constant interest rate.

```
nPer (rate, pmt, pv [ ,fv [ , type ] ])
```

Pmt

This function returns the payment for a loan based on periodic, constant payments and a constant interest rate. The result has a default number format of **money**.

```
Pmt (rate, nper, pv [ ,fv [ , type ] ] )
```

PV

This function returns the present value of an investment. The result has a default number format of **money**.

```
PV (rate, nper, pmt [ ,fv [ , type ] ])
```

Rate

This function returns the interest rate per period on annuity. The result has a default number format of **Fix** two decimals and %.

```
Rate (nper, pmt , pv [ ,fv [ , type ] ])
```

Black and Scholes

The Black and Scholes model is a mathematical model for financial market derivative instruments. The formula calculates the theoretical value of an option. In Qlik Sense, the **BlackAndSchole** function returns the value according to the Black and Scholes unmodified formula (European style options).

```
BlackAndSchole(strike , time_left , underlying_price , vol , risk_free_rate  
, type)
```

Strike	the future purchase price of the stock.
Time_left	the number of time periods remaining.
Underlying_price	the current value of the stock.
Vol	the volatility in % per time period.
Risk_free_rate	the risk free rate in % per time period.
Type	'c', 'call' or any non-zero numeric value for call-options and 'p', 'put' or 0 for put-options.

Example:

```
BlackAndSchole(130, 4, 68.5, 0.4, 0.04, 'call')  
returns 11.245...
```

5 Functions in scripts and chart expressions

(This is the theoretical price of an option to buy in 4 years at a value of 130 a share which is today worth 68.5 assuming a volatility of 40% per year and a risk-free interest rate of 4%)

FV

This function returns the future value of an investment based on periodic, constant payments and a constant interest rate. The result has a default number format of **money**.

Syntax:

```
FV(rate, nper, pmt [ ,pv [ , type ] ])
```

Arguments:

Argument	Description
rate	The interest rate per period.
nper	The total number of payment periods in an annuity.
pmt	The payment made each period. It cannot change over the life of the annuity. If pmt is omitted, the pv argument must be included.
pv	The present value, or lump-sum amount, that a series of future payments is worth right now. If pv is omitted, it is assumed to be 0 (zero) and the pmt argument must be included.
fv	The future value, or cash balance, you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0.
type	Should be 0 if payments are due at the end of the period and 1 if payments are due at the beginning of the period. If type is omitted, it is assumed to be 0.

Examples and results:

Example	Result
You are paying a new VCR by 36 monthly installments of \$20. The interest rate is 6% per annum. The bill comes at the end of every month. What is the total value of the money invested, when the last bill has been paid? <code>FV(0.005, 36, -20)</code>	Returns \$786.72

nPer

This function returns the number of periods for an investment based on periodic, constant payments and a constant interest rate.

Syntax:

```
nPer(rate, pmt, pv [ ,fv [ , type ] ])
```

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
rate	The interest rate per period.
nper	The total number of payment periods in an annuity.
pmt	The payment made each period. It cannot change over the life of the annuity. If pmt is omitted, the pv argument must be included.
pv	The present value, or lump-sum amount, that a series of future payments is worth right now. If pv is omitted, it is assumed to be 0 (zero) and the pmt argument must be included.
fv	The future value, or cash balance, you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0.
type	Should be 0 if payments are due at the end of the period and 1 if payments are due at the beginning of the period. If type is omitted, it is assumed to be 0.

Examples and results:

Example	Result
You want to sell a VCR by monthly installments of \$20. The interest rate is 6% per annum. The bill comes at the end of every month. How many periods are required if the value of the money received after the last bill has been paid should equal \$786.72? <code>nPer(0.005, -20, 0, 800)</code>	Returns 36

Pmt

This function returns the payment for a loan based on periodic, constant payments and a constant interest rate. The result has a default number format of **money**.

```
Pmt(rate, nper, pv [ ,fv [ , type ] ] )
```

To find the total amount paid over the duration of the loan, multiply the returned **pmt** value by **nper**.

Arguments:

Argument	Description
rate	The interest rate per period.
nper	The total number of payment periods in an annuity.
pmt	The payment made each period. It cannot change over the life of the annuity. If pmt is omitted, the pv argument must be included.
pv	The present value, or lump-sum amount, that a series of future payments is worth right now. If pv is omitted, it is assumed to be 0 (zero) and the pmt argument must be included.
fv	The future value, or cash balance, you want to attain after the last payment is made. If fv is

5 Functions in scripts and chart expressions

Argument	Description
	omitted, it is assumed to be 0.
type	Should be 0 if payments are due at the end of the period and 1 if payments are due at the beginning of the period. If type is omitted, it is assumed to be 0.

Examples and results:

Example	Result
The following formula returns the monthly payment on a \$20,000 loan at an annual rate of 10 percent, that must be paid off in 8 months: <code>Pmt(0.1/12,8,20000)</code>	Returns - \$2,594.66
For the same loan, if payment is due at the beginning of the period, the payment is: <code>Pmt(0.1/12,8,20000,0,1)</code>	Returns - \$2,573.21

PV

This function returns the present value of an investment. The result has a default number format of **money**.

```
PV(rate, nper, pmt [ ,fv [ , type ] ])
```

The present value is the total amount that a series of future payments is worth right now. For example, when borrowing money, the loan amount is the present value to the lender.

Arguments:

Argument	Description
rate	The interest rate per period.
nper	The total number of payment periods in an annuity.
pmt	The payment made each period. It cannot change over the life of the annuity. If pmt is omitted, the pv argument must be included.
pv	The present value, or lump-sum amount, that a series of future payments is worth right now. If pv is omitted, it is assumed to be 0 (zero) and the pmt argument must be included.
fv	The future value, or cash balance, you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0.
type	Should be 0 if payments are due at the end of the period and 1 if payments are due at the beginning of the period. If type is omitted, it is assumed to be 0.

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
What is the present value of \$100 paid to you at the end of each month during a 5 year period, given an interest rate of 7%? <code>PV(0.07/12,12*5,100,0,0)</code>	Returns - \$5,050.20

Rate

This function returns the interest rate per period on annuity. The result has a default number format of **Fix** two decimals and %.

Syntax:

```
Rate(nper, pmt , pv [ ,fv [ , type ] ])
```

The **rate** is calculated by iteration and can have zero or more solutions. If the successive results of **rate** do not converge, a NULL value will be returned.

Arguments:

Argument	Description
rate	The interest rate per period.
nper	The total number of payment periods in an annuity.
pmt	The payment made each period. It cannot change over the life of the annuity. If pmt is omitted, the pv argument must be included.
pv	The present value, or lump-sum amount, that a series of future payments is worth right now. If pv is omitted, it is assumed to be 0 (zero) and the pmt argument must be included.
fv	The future value, or cash balance, you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0.
type	Should be 0 if payments are due at the end of the period and 1 if payments are due at the beginning of the period. If type is omitted, it is assumed to be 0.

Examples and results:

Example	Result
What is the interest rate of a five-year \$10,000 annuity loan with monthly payments of \$300? <code>Rate(60,-300,10000)</code>	Returns 2.18%

5.10 Formatting functions

The formatting functions determine the display format of fields or expressions. With these functions it is possible to set decimal separator, thousands separator, and so on. The functions can be used both in data load scripts and chart expressions.



For reasons of clarity, all number representations are given with decimal point as decimal separator.

Formatting functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Date

The **date** function formats the expression as a date according to the given **format-code** string.

```
Date (expression [ , format-code ])
```

Dual

With the **dual** function it is possible to create arbitrary combinations of a number and a string.

```
Dual ( s , x )
```

Interval

This script function formats the **expression** as a time interval according to the string given as a **format-code**. If the format code is omitted, the time format set in the operating system is used.

```
Interval (expression [ , format-code ])
```

Money

The **money** function formats the **expression** numerically according to the string given as **format-code**. Decimal separator and thousands separator can be set as third and fourth parameters. If the parameters 2-4 are omitted, the number format that is set in the operating system will be used.

```
Money (expression [ , format-code [ , decimal-sep [ , thousands-sep ] ] )
```

Num

This script function formats the **expression** numerically according to the string given as **format-code**. Decimal separator and thousands separator can be set as third and fourth parameters. If the parameters 2-4 are omitted, the number format set in the operating system is used.

```
Num (expression [ , format-code [ , decimal-sep [ , thousands-sep ] ] )
```

Time

The time function formats the expression as time according to the string given as format-code. If the format

5 Functions in scripts and chart expressions

code is omitted, the time format set in the operating system is used.

```
Time (expression [ , format-code ])
```

Timestamp

This function formats the **expression** as a date and time according to the string given as **format-code**. If the format code is omitted, the date and time formats set in the operating system are used.

```
Timestamp (expression [ , format-code ])
```

Date

The **date** function formats the expression as a date according to the given **format-code** string.

Syntax:

```
Date (expression [ , format-code ])
```

Another way to describe it, is to look at it as a number to string conversion. The function takes the numeric value of the expression and generates a string representing the date according to the format code. The function returns a dual value with both the string and the number.

Arguments:

Argument	Description
format-code	String describing the format of the resulting string. If the format code is omitted, the date format set in the operating system is used.

Examples and results:

The examples below assume the two following operating system settings:

	Default setting 1	Default setting 2
Date format	YY-MM-DD	M/D/YY

Example	Results	Setting 1	Setting 2
Date(A) where A=35648	String:	97-08-06	8/6/97
	Number:	35648	35648
Date(A, 'YY.MM.DD') where A=35648	String:	97-08-06	8/6/97
	Number:	35648	35648

5 Functions in scripts and chart expressions

Example	Results	Setting 1	Setting 2
Date(A, 'DD.MM.YY') where A=35648.375	String:	06.08.1997	06.08.1997
	Number:	35648.375	35648.375
Date(A, 'YY.MM.DD') where A=8/6/97	String:	NULL (nothing)	97.08.06
	Number:	NULL	35648

Dual

With the **dual** function it is possible to create arbitrary combinations of a number and a string.

Syntax:

```
Dual ( s , x )
```

Forced association of an arbitrary string representation **s** with a given number representation **x**.

In Qlik Sense, all field values are potentially dual values. This means that the field values can have both a numeric value and a textual value. An example is a date that could have a numeric value of 40908 and the textual representation '2011-12-31'.

When several data items read into one field have different string representations but the same valid number representation, they will all share the first string representation encountered.



*The **dual** function is typically used early in the script, before other data is read into the field concerned, in order to create that first string representation, which will be shown in filter panes.*

Examples and results:

Example	Description
Dual('Q' & Ceil(Month(Date)/3), Ceil(Month(Date)/3)) as Quarter	This field definition will create a field Quarter with the textual values 'Q1' to 'Q4' and at the same time assign the numeric values 1 to 4.
Dual(WeekYear(Date) & '-W' & Week(Date), WeekStart(Date)) as YearWeek	This field definition will create a field YearWeek with textual values of the form '2012-W22' and at the same time assign a numeric value corresponding to the date number of the first day of the week, e.g. 41057.

Interval

This script function formats the **expression** as a time interval according to the string given as a **format-code**. If the format code is omitted, the time format set in the operating system is used.

Syntax:

5 Functions in scripts and chart expressions

Interval(expression [, format-code])

Intervals may be formatted as a time, as days or as a combination of days, hours, minutes, seconds and fractions of seconds.

Examples and results:

The examples below assume the following operating system settings:

Short date format: YY-MM-DD

Time format: hh:mm:ss

Number decimal separator: .

Example	String	Number
Interval(A) where A=0.37	09:00:00	0.375
Interval(A) where A=1.375	33:00:00	1.375
Interval(A, 'D hh:mm') where A=1.375	1 09:00	1.375
Interval(A-B, 'D hh:mm') where A=97-08-06 09:00:00 and B=96-08-06 00:00:00	365 09:00	365.375

Num

This script function formats the **expression** numerically according to the string given as **format-code**.

Decimal separator and thousands separator can be set as third and fourth parameters. If the parameters 2-4 are omitted, the number format set in the operating system is used.

Syntax:

Num(expression [, format-code [, decimal-sep [, thousands-sep]]])

Examples and results:

The examples below assume the following operating system settings:

	Default setting 1	Default setting 2
Number format	# ##0,#	#,##0.#

Example	Results	Setting 1	Setting 2
Num(A, '0.0') where A=35648.375	String:	35 648 375	35648.375
	Number:	35648375	35648.375

5 Functions in scripts and chart expressions

Example	Results	Setting 1	Setting 2
Num(A, '#,##0.##', '.' , ',') where A=35648	String:	35,648.00	35,648.00
	Number:	35648	35648
Num(pi(), '0,00')	String:	3,14	003
	Number:	3.141592653	3.141592653

Money

The **money** function formats the **expression** numerically according to the string given as **format-code**.

Decimal separator and thousands separator can be set as third and fourth parameters. If the parameters 2-4 are omitted, the number format that is set in the operating system will be used.

Syntax:

```
Money ( expression [ , format-code [ , decimal-sep [ , thousands-sep ] ] ] )
```

Examples and results:

The examples below assume the following operating system settings:

	Default setting 1	Default setting 2
Money format	kr # ##0,00	\$ #,##0.00

Example	Results	Setting 1	Setting 2
Money(A) where A=35648	String:	kr 35 648,00	\$ 35,648.00
	Number:	35648.00	35648.00
Money(A, '#,##0 ¥', '.' , ',') where A=3564800	String:	3,564,800 ¥	3,564,800 ¥
	Number:	3564800	3564800

Time

The time function formats the expression as time according to the string given as format-code. If the format code is omitted, the time format set in the operating system is used.

Syntax:

```
Time ( expression [ , format-code ] )
```

Examples and results:

The examples below assume the following operating system settings:

5 Functions in scripts and chart expressions

	Default setting 1	Default setting 2
Time format	hh:mm:ss	hh.mm.ss

Example	Results	Setting 1	Setting 2
Time(A) where A=0.375	String:	09:00:00	09.00.00
	Number:	0.375	0.375
Time(A) where A=35648.375	String:	09:00:00	09.00.00
	Number:	35648.375	35648.375
Time(A, 'hh-mm') where A=0.99999	String:	23-59	23-59
	Number:	0.99999	0.99999

Timestamp

This function formats the **expression** as a date and time according to the string given as **format-code**. If the format code is omitted, the date and time formats set in the operating system are used.

Syntax:

```
Timestamp(expression [ , format-code ])
```

Examples and results:

The examples below assume the following operating system settings:

	Default setting 1	Default setting 2
Timestamp format	YY-MM-DD	M/D/YY

Example	Results	Setting 1	Setting 2
Timestamp(A) where A=35648.375	String:	97-08-06 09:00:00	8/6/97 09:00:00
	Number:	35648.375	35648.375
Timestamp(A, 'YYYY-MM-DD hh.mm') where A=35648	String:	1997-08-06 00.00	1997-08-06 00.00
	Number:	35648	35648

5.11 General numeric functions in charts

In these general numeric functions, the arguments are expressions where **x** should be interpreted as a real valued number. All functions can be used in both data load scripts and chart expressions.

5 Functions in scripts and chart expressions

bitcount

BitCount() finds how many bits in the binary equivalent of a number are set to 1. That is, the function returns the number of set bits in **integer_number**, where **integer_number** is interpreted as a signed 32-bit integer.

```
BitCount (integer_number)
```

ceil

Ceil() rounds up a number to the nearest multiple of the specified **step** interval. The result is increased by the value of **offset**, if one is specified, or reduced if **offset** is negative.

```
Ceil (x[, step[, offset]])
```

combin

Combin() returns the number of combinations of **q** elements that can be picked from a set of **p** items. As represented by the formula: $\text{combin}(p,q) = p! / q!(p-q)!$ The order in which the items are selected is insignificant.

```
Combinp, q)
```

div

Div() returns the integer part of the arithmetic division of the first argument by the second argument. Both parameters are interpreted as real numbers, that is, they do not have to be integers.

```
Div(integer_number1, integer_number2)
```

even

Even() returns True (-1), if **integer_number** is an even integer or zero. It returns False (0), if **integer_number** is an odd integer, and NULL if **integer_number** is not an integer.

```
Even (integer_number)
```

fabs

Fabs() returns the absolute value of **x**. The result is a positive number.

```
Fabs (x)
```

fact

Fact() returns the factorial of a positive integer **x**.

```
Fact (x)
```

floor

Floor() rounds down a number to the nearest multiple of the specified **step** interval. The result is increased by the value of **offset**, if one is specified, or reduced if **offset** is negative.

```
Floor (x[, step[, offset]])
```

fmod

fmod() is a generalized modulo function that returns the remainder part of the integer division of the first

5 Functions in scripts and chart expressions

argument (the dividend) by the second argument (the divisor). The result is a real number. Both arguments are interpreted as real numbers, that is, they do not have to be integers.

Fmod(a, b)

frac

Frac() returns the fraction part of **x**.

Frac(x)

mod

Mod() is a mathematical modulo function that returns the non-negative remainder of an integer division. The first argument is the dividend, the second argument is the divisor, Both arguments must be integer values.

Mod(integer_number1, integer_number2)

odd

Odd() returns True (-1), if **integer_number** is an odd integer or zero. It returns False (0), if **integer_number** is an even integer, and NULL if **integer_number** is not an integer.

Odd(integer_number)

permut

Permut() returns the number of permutations of **q** elements that can be selected from a set of **p** items. As represented by the formula: $\text{Permut}(p,q) = \frac{p!}{(p - q)!}$ The order in which the items are selected is significant.

Permut(p, q)

round

Round() returns the result of rounding **x** up or down to the nearest multiple of **step**. The result is increased by the value of **offset**, if one is specified, or reduced if **offset** is negative.

Round(x [, base [, offset]])

sign

Sign() returns 1, 0 or -1 depending on whether **x** is a positive number, 0, or a negative number.

Sign(x)

BitCount

BitCount() finds how many bits in the binary equivalent of a number are set to 1. That is, the function returns the number of set bits in **integer_number**, where **integer_number** is interpreted as a signed 32-bit integer.

Syntax:

BitCount(integer_number)

Return data type: integer

Examples and results:

Examples	Results
BitCount (3)	3 is binary 101, therefore this returns 2
BitCount (-1)	-1 is 32 ones in binary, therefore this returns 32

Ceil

Ceil() rounds up a number to the nearest multiple of the specified **step** interval. The result is increased by the value of **offset**, if one is specified, or reduced if **offset** is negative.

Compare with the **floor** function, which rounds input numbers down.

Syntax:

```
Ceil(x[, step[, offset]])
```

Return data type: integer

Examples and results:

Examples	Results
ceil(2.4)	Returns 4
ceil(2.6)	Returns 6
ceil(3.88 , 0.1)	Returns 3.9
ceil(3.88 , 5)	Returns 5
ceil(1.1 , 1)	Returns 2
ceil(1.1 , 1 , 0.5)	Returns 1.5
ceil(1.1 , 1 , -0.01)	Returns 1.99

See also:

▫ [Floor \(page 434\)](#)

Combin

Combin() returns the number of combinations of **q** elements that can be picked from a set of **p** items. As represented by the formula: $\text{combin}(p,q) = p! / q!(p-q)!$ The order in which the items are selected is insignificant.

Syntax:

5 Functions in scripts and chart expressions

Combin(p, q)

Return data type: integer

Limitations:

Non-integer items will be truncated.

Examples and results:

Examples	Results
How many combinations of 7 numbers can be picked from a total of 35 lottery numbers? <code>Combin(35,7)</code>	Returns 6,724,520

See also:

▢ [Permut \(page 437\)](#)

Div

Div() returns the integer part of the arithmetic division of the first argument by the second argument. Both parameters are interpreted as real numbers, that is, they do not have to be integers.

Syntax:

Div(integer_number1, integer_number2)

Return data type: integer

Examples and results:

Examples	Results
<code>Div(7,2)</code>	Returns 3
<code>Div(7.1,2.3)</code>	Returns 3
<code>Div(9,3)</code>	Returns 3
<code>Div(-4,3)</code>	Returns -1
<code>Div(4,-3)</code>	Returns -1
<code>Div(-4,-3)</code>	Returns 1

See also:

▢ [Mod \(page 436\)](#)

Even

Even() returns True (-1), if **integer_number** is an even integer or zero. It returns False (0), if **integer_number** is an odd integer, and NULL if **integer_number** is not an integer.

Syntax:

```
Even (integer_number)
```

Return data type: integer

Examples and results:

Examples	Results
Even(3)	Returns 0, False
Even(2 * 10)	Returns -1, True
Even(3.14)	Returns NULL

See also:

▢ [Odd \(page 437\)](#)

Fabs

Fabs() returns the absolute value of **x**. The result is a positive number.

Syntax:

```
fabs (x)
```

Return data type: numeric

Examples and results:

Examples	Results
fabs(2.4)	Returns 2.4
fabs(-3.8)	Returns 3.8

Fact

Fact() returns the factorial of a positive integer **x**.

Syntax:

```
Fact(x)
```

Return data type: integer

Limitations:

If the number **x** is not an integer, it will be truncated. Non-positive numbers will return NULL.

Examples and results:

Examples	Results
Fact(1)	Returns 1
Fact(5)	Returns 120 (1 * 2 * 3 * 4 * 5 = 120)
Fact(-5)	Returns NULL

Floor

Floor() rounds down a number to the nearest multiple of the specified **step** interval. The result is decreased by the value of **offset**, if one is specified, or increased if **offset** is negative.

Compare with the **ceil** function, which rounds input numbers up.

Syntax:

```
Floor(x[, step[, offset]])
```

Return data type: numeric

Examples and results:

Examples	Results
Floor(2,4)	Returns 0
Floor(4,2)	Returns 4
Floor(3.88 , 0.1)	Returns 3.8
Floor(3.88 , 5)	Returns 0
Floor(1.1 , 1)	Returns 1
Floor(1.1 , 1 , 0.5)	Returns 0.5

See also:

- ▢ [Frac \(page 435\)](#)
- ▢ [Ceil \(page 431\)](#)

Fmod

fmod() is a generalized modulo function that returns the remainder part of the integer division of the first argument (the dividend) by the second argument (the divisor). The result is a real number. Both arguments are interpreted as real numbers, that is, they do not have to be integers.

Syntax:

```
fmod( a, b )
```

Return data type: numeric

Examples and results:

Examples	Results
<code>fmod(7,2)</code>	Returns 1
<code>fmod(7.5,2)</code>	Returns 1.5
<code>fmod(9,3)</code>	Returns 0
<code>fmod(-4,3)</code>	Returns -1
<code>fmod(4,-3)</code>	Returns 1
<code>fmod(-4,-3)</code>	Returns -1

See also:

- ▢ [Div \(page 432\)](#)

Frac

Frac() returns the fraction part of **x**.

The fraction is defined in such a way that $\text{Frac}(x) + \text{Floor}(x) = x$. In simple terms this means that the fractional part of a positive number is the difference between the number (x) and the integer that precedes it.

For example:

The fractional part of 11.43 = 11.43 - 11 = 0.43

For a negative number, say -1.4, $\text{Floor}(-1.4) = -2$, which produces the following result:

The fractional part of -1.4 = 1.4 - (-2) = -1.4 + 2 = 0.6

Syntax:

```
Frac(x)
```

Return data type: numeric

Examples and results:

Examples	Results
Frac(11.43)	Returns 0.43
Frac(-1.4)	Returns 0.6

See also:

▢ [Floor \(page 434\)](#)

Mod

Mod() is a mathematical modulo function that returns the non-negative remainder of an integer division. The first argument is the dividend, the second argument is the divisor, Both arguments must be integer values.

Syntax:

```
Mod(integer_number1, integer_number2)
```

Return data type: integer

Limitations:

integer_number2 must be greater than 0.

Examples and results:

Examples	Results
Mod(7,2)	Returns 1
Mod(7.5,2)	Returns NULL
Mod(9,3)	Returns 0
Mod(-4,3)	Returns 2
Mod(4,-3)	Returns NULL
Mod(-4,-3)	Returns NULL

See also:

▢ [Div \(page 432\)](#)

Odd

Odd() returns True (-1), if **integer_number** is an odd integer or zero. It returns False (0), if **integer_number** is an even integer, and NULL if **integer_number** is not an integer.

Syntax:

```
Odd(integer_number)
```

Return data type: integer

Examples and results:

Examples	Results
odd(3)	Returns -1, True
odd(2 * 10)	Returns 0, False
odd(3.14)	Returns NULL

See also:

▢ [Even \(page 433\)](#)

Permut

Permut() returns the number of permutations of **q** elements that can be selected from a set of **p** items. As represented by the formula: $Permut(p,q) = (p)! / (p - q)!$ The order in which the items are selected is significant.

Syntax:

```
Permut (p, q)
```

Return data type: integer

Limitations:

Non-integer arguments will be truncated.

Examples and results:

5 Functions in scripts and chart expressions

Examples	Results
In how many ways could the gold, silver and bronze medals be distributed after a 100 m final with 8 participants? <code>Permut(8,3)</code>	Returns 336

See also:

▢ [Combin \(page 431\)](#)

Round

Round() returns the result of rounding **x** up or down to the nearest multiple of **step**. . The result is increased by the value of **offset**, if one is specified, or reduced if **offset** is negative. The default value of **step** is 1.

If **x** is exactly in the middle of an interval, it is rounded upwards.

Syntax:

```
Round(x[, step[, offset]])
```

Return data type: numeric

Examples and results:

Examples	Results
<code>Round(3.8)</code>	Returns 4
<code>Round(3.8.4)</code>	Returns 4
<code>Round(2.5)</code>	Returns 3. Rounded up because 2.5 is exactly half of the default step interval.
<code>Round(2.4)</code>	Returns 4. Rounded up because 2 is exactly half of the step interval of 4.
<code>Round(2.6)</code>	Returns 0
<code>Round(3.88 , 0.1)</code>	Returns 3.9
<code>Round(3.88 , 5)</code>	Returns 5
<code>Round(1.1 , 1 , 0.5)</code>	Returns 1.5

Sign

Sign() returns 1, 0 or -1 depending on whether **x** is a positive number, 0, or a negative number.

Syntax:

```
Sign(x)
```

Return data type: numeric

Limitations:

If no numeric value is found, NULL is returned.

Examples and results:

Examples	Results
sign(66)	Returns 1
sign(0)	Returns 0
sign(- 234)	Returns -1

5.12 Geographical functions

These functions are used to handle geographical data in map visualizations.

Geographical functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Parameters used in geographical functions

Geometry	This can be any of the following: <ul style="list-style-type: none">• a point (latitude, longitude)• an area
Projection	With Mercator projection you can represent maps in square format, correcting for the distortion created by stretching. This can be any of the following: <ul style="list-style-type: none">• 'unit' (default) - projection is 1:1• 'mercator'

GeoAggrGeometry

This function can be used to aggregate a number of areas into a larger area, for example aggregating a number of sub-regions to a region.

```
GeoAggrGeometry(geometry)
```

GeoBoundingBox

This function can be used in scripts to aggregate a geometry into an area and calculate the smallest GeoBoundingBox that contain all coordinates.

5 Functions in scripts and chart expressions

A `GeoBoundingBox` is represented as a list of four values, left, right, top, bottom.

```
GeoBoundingBox(geometry)
```

GeoGetBoundingBox

This function can be used in scripts and chart expressions to calculate the smallest `GeoBoundingBox` that contain all coordinates of a geometry.

A `GeoBoundingBox` is represented as a list of four values, left, right, top, bottom.

```
GeoGetBoundingBox(geometry)
```

GeoGetPolygonCenter

This function can be used in scripts and chart expressions to calculate and return the center point of a geometry.

```
GeoGetPolygonCenter(geometry)
```

GeoInvProjectGeometry

This function can be used in scripts to aggregate a geometry into an area and apply the inverse of a projection.

```
GeoInvProjectGeometry(projection, geometry)
```

GeoMakePoint

This function can be used in scripts and chart expressions to create and tag a `Point` with latitude and longitude.

```
GeoMakePoint(latitude, longitude )
```

GeoProject

This function can be used in scripts and chart expressions to apply a projection to a geometry.

```
GeoProject(projection, geometry)
```

GeoProjectGeometry

This function can be used in scripts to aggregate a geometry into an area and apply a projection.

```
GeoProjectGeometry(projection, geometry)
```

GeoReduceGeometry

This function can be used in scripts to aggregate a geometry into an area.

```
GeoReduceGeometry(geometry)
```

5.13 Interpretation functions

The interpretation functions interpret the contents of fields or expressions. With these functions it is possible to state the data type, decimal separator, thousands separator etc. used. All functions can be used in both data load scripts and chart expressions.

5 Functions in scripts and chart expressions

If no interpretation functions are used, Qlik Sense interprets the data as a mix of numbers, dates, times, time stamps and strings, using the default settings for number format, date format and time format defined by script variables and by the operating system.



For reasons of clarity, all number representations are given with decimal point as decimal separator.

Interpretation functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Date#

This function evaluates the expression as a date according to the string given as **format-code**. If the format code is omitted, the default date format set in the operating system is used.

```
Date# (expression [ , format-code ])
```

Interval#

This function evaluates the **expression** as a time interval according to the string given as a format-code. If the **format code** is omitted, the time format set in the operating system is used.

```
Interval# (expression [ , format-code ])
```

Money#

This function evaluates the **expression** numerically according to the string given as **format-code**. Decimal separator and thousands separator can be set as third and fourth parameters. If the parameters 2-4 are omitted, the default number format set by script variables or in the operating system is used.

```
Money# (expression [ , format-code [ , decimal-sep [ , thousands-sep ] ] )
```

Num#

This function evaluates the expression numerically according to the string given as format-code. Decimal separator and thousands separator can be set as third and fourth parameters. If the parameters 2-4 are omitted, the default number format set by script variables or in the operating system is used.

```
Num# (expression [ , format-code [ , decimal-sep [ , thousands-sep ] ] )
```

Text

The **text** function forces the expression to be treated as text, even if a numeric interpretation is possible.

```
Text (expression )
```

Time#

The **time#** function evaluates the **expression** as time according to the string given as **format-code**. If the format code is omitted, the default time format set in the operating system is used.

5 Functions in scripts and chart expressions

```
Time# (expression [ , format-code ])
```

Timestamp#

The **timestamp#** function evaluates the **expression** as a date and time according to the string given as **format-code**. If the **format code** is omitted, the default date and time formats set in the operating system are used.

```
Timestamp#(expression [ , format-code ])
```

Date#

This function evaluates the expression as a date according to the string given as **format-code**. If the format code is omitted, the default date format set in the operating system is used.

Syntax:

```
Date#(expression [ , format-code ])
```

Arguments:

Argument	Description
format-code	String describing the format of the resulting string. If the format code is omitted, the date format set in the operating system is used.

Examples and results:

The examples below assume the two following operating system settings:

	Default setting 1	Default setting 2
Date format	YY-MM-DD	M/D/YY

Example	Results	Result with Setting 1	Result with Setting 2
Date#(A) where A=8/6/97	String:	8/6/97	8/6/97
	Number:	-	35648
Date#(A, 'YYYY.MM.DD') where A=1997.08.06	String:	1997.08.08	1997.08.06
	Number:	35648	35648

Interval#

This function evaluates the **expression** as a time interval according to the string given as a **format-code**. If the **format code** is omitted, the time format set in the operating system is used.

Syntax:

```
Interval#(expression [ , format-code ])
```

5 Functions in scripts and chart expressions

The **interval#** function generally behaves just like the **time#** function but while times can never be greater than 23:59:59 (numeric value 0.99999) or smaller than 00:00:00 (numeric value 0.00000) an interval may have any value.

Examples and results:

The examples below assume the following operating system settings:

Short date format: YY-MM-DD

Time format: hh:mm:ss

Number decimal separator: .

Example		Result
Interval#(A, 'D hh:mm') where A=1 09:00	String:	1 09:00
	Number:	1.375
Interval#(A-B) where A=97-08-06 09:00:00 and B=97-08-05 00:00:00	String:	1.375
	Number:	1.375

Money#

This function evaluates the **expression** numerically according to the string given as **format-code**. Decimal separator and thousands separator can be set as third and fourth parameters. If the parameters 2-4 are omitted, the default number format set by script variables or in the operating system is used.

Syntax:

```
Money# (expression [ , format-code [ , decimal-sep [ , thousands-sep ] ] )
```

The **money#** function generally behaves just like the **num#** function but takes its default values for decimal and thousand separator from the script variables for money format or the system settings for currency.

Examples and results:

The examples below assume the two following operating system settings:

	Default setting 1	Default setting 2
Money format	kr # ##0,00	\$ #,##0.00

5 Functions in scripts and chart expressions

Example	Results	Setting 1	Setting 2
Money#(, '# ##0,00 kr') where A=35 648,37 kr	String:	35 648.37 kr	35 648.37 kr
	Number:	35648.37	3564837
Money#(A, '\$#', '.', ',') where A= \$35,648.37	String:	\$35,648.37	\$35,648.37
	Number:	35648.37	35648.37

Num#

This function evaluates the expression numerically according to the string given as format-code. Decimal separator and thousands separator can be set as third and fourth parameters. If the parameters 2-4 are omitted, the default number format set by script variables or in the operating system is used.

Syntax:

```
Num# (expression [ , format-code[ , decimal-sep [ , thousands-sep] ] ] )
```

Examples and results:

The examples below assume the two following operating system settings:

	Default setting 1	Default setting 2
Number format	# ##0,#	#,##0.#

Example	Results	Setting 1	Setting 2
Num#(A, '#') where A=35,648.375	String:	35,648.375	35648.375
	Number:	-	35648.375
Num#(A, '#.#', '.', ',') where A=35,648.375	String:	35,648.375	35,648.375
	Number:	35648.375	35648.375
Num#(A, '#.#', ',', '.') where A=35648.375	String:	35648.375	35648.375
	Number:	35648375	35648375
Num#(A, 'abc#,#') where A=abc123,4	String:	abc123,4	abc123,4
	Number:	123.4	1234

Text

The **text** function forces the expression to be treated as text, even if a numeric interpretation is possible.

Syntax:

```
Text (expression )
```

5 Functions in scripts and chart expressions

Examples and results:

Example		Result
Text(A) where A=1234	String:	1234
	Number:	-
Text(pi())	String:	3.1415926535898
	Number:	-

Time#

The **time#** function evaluates the **expression** as time according to the string given as **format-code**. If the format code is omitted, the default time format set in the operating system is used.

Syntax:

```
time#(expression [ , format-code ])
```

Examples and results:

The examples below assume the two following operating system settings:

	Default setting 1	Default setting 2
Time format	hh:mm:ss	hh.mm.ss

Example	Results	Setting 1	Setting 2
time#(A) where A=09:00:00	String:	09:00:00	09:00:00
	Number:	0.375	-
time#(A, 'hh.mm') where A=09.00	String:	09.00	09.00
	Number:	0.375	0.375

Timestamp#

The **timestamp#** function evaluates the **expression** as a date and time according to the string given as **format-code**. If the **format code** is omitted, the default date and time formats set in the operating system are used.

Syntax:

```
timestamp#(expression [ , format-code ])
```

Examples and results:

The examples below assume the two following operating system settings:

5 Functions in scripts and chart expressions

	Default setting 1	Default setting 2
Date format	YY-MM-DD	M/D/YY
Time format	hh:mm:ss	hh:mm:ss

Example	Results	Setting 1	Setting 2
timestamp#(A) where A=8/6/97 09:00:00	String:	8/6/97 09:00:00	8/6/97 09:00:00
	Number:	-	35648.375
timestamp#(A, 'YYYY-MM-DD hh_mm') where A=8/6/97 09_00	String:	1997-08-06 09_00	1997-08-06 09_00
	Number:	35648.375	35648.375

5.14 Inter-record functions

Inter-record functions are used:

- In the data load script, when a value from previously loaded records of data is needed for the evaluation of the current record.
- In a chart expression, when another value from the data set of a visualization is needed.



Sorting on y-values in charts or sorting by expression columns in straight tables is not allowed when chart inter-record functions are used in any of the chart's expressions. These sort alternatives are therefore automatically disabled.

Suppression of zero values is automatically disabled when these functions are used.

Row functions

These functions can only be used in chart expressions.

Above

Above() evaluates an expression at a row above the current row within a column segment in a table. The row for which it is calculated depends on the value of **offset**, if present, the default being the row directly above. For charts other than tables, **Above()** evaluates for the row above the current row in the chart's straight table equivalent.

```
Above - chart function([TOTAL [<fld{,fld}>]] expr [ , offset [,count]])
```

Below

Below() evaluates an expression at a row below the current row within a column segment in a table. The row for which it is calculated depends on the value of **offset**, if present, the default being the row directly below. For charts other than tables, **Below()** evaluates for the row below the current column in the chart's straight

5 Functions in scripts and chart expressions

table equivalent.

```
Below - chart function([TOTAL [<fld{,fld}>]] expression [ , offset [,count  
]])
```

Bottom

Bottom() evaluates an expression at the last (bottom) row of a column segment in a table. The row for which it is calculated depends on the value of **offset**, if present, the default being the bottom row. For charts other than tables, the evaluation is made on the last row of the current column in the chart's straight table equivalent.

```
Bottom - chart function([TOTAL [<fld{,fld}>]] expr [ , offset [,count ]])
```

Top

Top() evaluates an expression at the first (top) row of a column segment in a table. The row for which it is calculated depends on the value of **offset**, if present, the default being the top row. For charts other than tables, the **Top()** evaluation is made on the first row of the current column in the chart's straight table equivalent.

```
Top - chart function([TOTAL [<fld{,fld}>]] expr [ , offset [,count ]])
```

NoOfRows

NoOfRows() returns the number of rows in the current column segment in a table. For bitmap charts, **NoOfRows()** returns the number of rows in the chart's straight table equivalent.

```
NoOfRows - chart function([TOTAL])
```

RowNo

RowNo() returns the number of the current row within the current column segment in a table. For bitmap charts, **RowNo()** returns the number of the current row within the chart's straight table equivalent.

```
RowNo - chart function([TOTAL])
```

Column functions

These functions can only be used in chart expressions.

Column

Column() returns the value found in the column corresponding to **ColumnNo** in a straight table, disregarding dimensions. For example **Column(2)** returns the value of the second measure column.

```
Column - chart function(ColumnNo)
```

Dimensionality

Dimensionality() returns the number of dimensions for the current row.

```
Dimensionality - chart function ( )
```

Field functions

FieldIndex

5 Functions in scripts and chart expressions

FieldIndex() returns the position of the field value **value** in the field **field_name** (by load order).

```
FieldIndex (field_name , value)
```

FieldValue

FieldValue() returns the value found in position **elem_no** of the field **field_name** (by load order).

```
FieldValue (field_name , elem_no)
```

FieldValueCount

FieldValueCount() is an **integer** function that finds the number of distinct values in a field.

```
FieldValueCount (field_name)
```

Inter-record functions in the data load script

Exists

This script function determines whether a specific field value exists in a specified field of the data loaded so far. **Field** is a name or a string expression evaluating to a field name.

```
Exists (field [ , expression ])
```

LookUp

This script function returns the value of **fieldname** corresponding to the first occurrence of the value **matchfieldvalue** in the field **matchfieldname**.

```
LookUp (fieldname, matchfieldname, matchfieldvalue [, tablename])
```

Peek

This script function returns the contents of the **fieldname** in the record specified by **row** in the internal table **tablename**. Data are fetched from the associative Qlik Sense database.

```
Peek (fieldname [ , row [ , tablename ] ])
```

Previous

This script function returns the value of **expression** using data from the previous input record that was not discarded due to a **where** clause. In the first record of an internal table the function will return NULL.

```
Previous (expression )
```

See also:

Range functions (page 476)

Above - chart function

Above() evaluates an expression at a row above the current row within a column segment in a table. The row for which it is calculated depends on the value of **offset**, if present, the default being the row directly above. For charts other than tables, **Above()** evaluates for the row above the current row in the chart's straight table equivalent.

Syntax:

```
Above ([TOTAL] expr [ , offset [,count]])
```

Return data type: dual

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
offset	Specifying an offset <i>n</i> , greater than 1 moves the evaluation of the expression <i>n</i> rows further up from the current row. Specifying an offset of 0 will evaluate the expression on the current row. Specifying a negative offset number makes the Above function work like the Below function with the corresponding positive offset number.
count	By specifying a third parameter count greater than 1, the function will return a range of count values, one for each of count table rows counting upwards from the original cell. In this form, the function can be used as an argument to any of the special range functions. <i>Range functions (page 476)</i>
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.

On the first row of a column segment, a NULL value is returned, as there is no row above it.



A column segment is defined as a consecutive subset of cells having the same values for the dimensions in the current sort order. Inter-record chart functions are computed in the column segment excluding the right-most dimension in the equivalent straight table chart. If there is only one dimension in the chart, or if the TOTAL qualifier is specified, the expression evaluates across full table.



If the table or table equivalent has multiple vertical dimensions, the current column segment will include only rows with the same values as the current row in all dimension columns, except for the column showing the last dimension in the inter-field sort order.

5 Functions in scripts and chart expressions

Limitations:

Recursive calls will return NULL.

Examples and results:

Example 1:

Customer	Sum(Sales)	Above(Sum(Sales))	Sum(Sales)+Above(Sum(Sales))	Above offset 3	Higher?
	2566	-	-	-	-
Astrida	587	-	-	-	-
Betacab	539	587	1126	-	-
Canutility	683	539	1222	-	Higher
Divadip	757	683	1440	1344	Higher

The table visualization for Example 1.

In the screenshot of the table shown in this example, the table visualization is created from the dimension **Customer** and the measures: `sum(Sales)` and `Above(Sum(Sales))`.

The column **Above(Sum(Sales))** returns NULL for the **Customer** row containing **Astrida**, because there is no row above it. The result for the row **Betacab** shows the value of `Sum(Sales)` for **Astrida**, the result for **Canutility** shows the value for `Sum(Sales)` for **Betacab**, and so on.

For the column labeled **Sum(Sales)+Above(Sum(Sales))**, the row for **Betacab** shows the result of the addition of the `Sum(Sales)` values for the rows **Betacab + Astrida** (539+587). The result for the row **Canutility** shows the result of the addition of `Sum(Sales)` values for **Canutility + Betacab** (683+539).

The measure labeled **Above offset 3** created using the expression `sum(Sales)+Above(Sum(Sales), 3)` has the argument **offset**, set to 3, and has the effect of taking the value in the row three rows above the current row. It adds the `Sum(Sales)` value for the current **Customer** to the value for the **Customer** three rows above. The values returned for the first three **Customer** rows are null.

The table also shows more complex measures: one created from `sum(Sales)+Above(Sum(Sales))` and one labeled **Higher?**, which is created from `IF(Sum(Sales)>Above(Sum(Sales)), 'Higher')`.



This function can also be used in charts other than tables, for example bar charts.



For other chart types, convert the chart to the straight table equivalent so you can easily interpret which row the function relates to.

Example 2:

5 Functions in scripts and chart expressions

In the screenshots of tables shown in this example, more dimensions have been added to the visualizations: **Month** and **Product**. For charts with more than one dimension, the results of expressions containing the **Above**, **Below**, **Top**, and **Bottom** functions depend on the order in which the column dimensions are sorted by Qlik Sense. Qlik Sense evaluates the functions based on the column segments that result from the dimension that is sorted last. The column sort order is controlled in the properties panel under **Sorting** and is not necessarily the order in which the columns appear in a table.

In the following screenshot of table visualization for Example 2, the last-sorted dimension is **Month**, so the **Above** function evaluates based on months. There is a series of results for each **Product** value for each month (**Jan** to **Aug**) - a column segment. This is followed by a series for the next column segment: for each **Month** for the next **Product**. There will be a column segment for each **Customer** value for each **Product**.

Customer	Product	Month	Sum([Sales])	Above(Sum(Sales))
			2566	-
Astrida	AA	Jan	46	-
Astrida	AA	Feb	60	46
Astrida	AA	Mar	70	60
Astrida	AA	Apr	13	70
Astrida	AA	May	78	13
Astrida	AA	Jun	20	78
Astrida	AA	Jul	45	20
Astrida	AA	Aug	65	45

The table visualization for Example 2.

Example 3:

In the screenshot of table visualization for Example 3, the last sorted dimension is **Product**. This is done by moving the dimension Product to position 3 in the Sorting tab in the properties panel. The **Above** function is evaluated for each **Product**, and because there are only two products, **AA** and **BB**, there is only one non-null result in each series. Row **BB** for **Jan** shows the value for **Sum(Sales)**, for row **AA**, the value is null. The value in each row **AA** will always be null, as there is no **AA** is the first value of **Product** in the column segment. The second series is evaluated on **AA** and **BB** for **Feb**, and so on for the first **Customer** value, **Astrida**. When all the months are exhausted for the first **Customer** value, the sequence is repeated for the second **Customer** value, and so on.

5 Functions in scripts and chart expressions

Customer	Product	Month	Sum([Sales])	Above(Sum(Sales))
			2566	-
Astrida	AA	Jan	46	-
Astrida	BB	Jan	46	46
Astrida	AA	Feb	60	-
Astrida	BB	Feb	60	60
Astrida	AA	Mar	70	-
Astrida	BB	Mar	70	70
Astrida	AA	Apr	13	-
Astrida	BB	Apr	13	13

The table visualization for Example 3.

Example 4:	Result
The Above function can be used as input to the range functions. For example: <code>RangeAvg (Above(Sum (Sales),1,3))</code> .	Returns the average of the three results of the Sum (Sales) function evaluated on the three rows immediately above the current row.

Data used in examples:

Monthnames:

```
LOAD * INLINE [
Month, Monthnumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];
```

Sales2013:

```
crosstable (Month, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

To get the months to sort in the correct order, when you create your visualizations, go to the **Sorting** section of the properties panel, select **Month** and mark the checkbox **Sort by expression**. In the expression box write Monthnumber.

See also:

- *Below - chart function (page 456)*
- *Bottom - chart function (page 453)*
- *Top - chart function (page 468)*
- *RangeAvg (page 478)*

Bottom - chart function

Bottom() evaluates an expression at the last (bottom) row of a column segment in a table. The row for which it is calculated depends on the value of **offset**, if present, the default being the bottom row. For charts other than tables, the evaluation is made on the last row of the current column in the chart's straight table equivalent.

Syntax:

```
Bottom([TOTAL] expr [ , offset [,count ]])
```

Return data type: dual

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
offset	Specifying an offset <i>n</i> greater than 1 moves the evaluation of the expression up <i>n</i> rows above the bottom row. Specifying a negative offset number makes the Bottom function work like the Top function with the corresponding positive offset number.
count	By specifying a third parameter count greater than 1, the function will return not one but a range of count values, one for each of the last count rows of the current column segment. In this form, the function can be used as an argument to any of the special range functions. <i>Range functions (page 476)</i>
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.

5 Functions in scripts and chart expressions



A column segment is defined as a consecutive subset of cells having the same values for the dimensions in the current sort order. Inter-record chart functions are computed in the column segment excluding the right-most dimension in the equivalent straight table chart. If there is only one dimension in the chart, or if the **TOTAL** qualifier is specified, the expression evaluates across full table.



If the table or table equivalent has multiple vertical dimensions, the current column segment will include only rows with the same values as the current row in all dimension columns, except for the column showing the last dimension in the inter-field sort order.

Limitations:

Recursive calls will return NULL.

Examples and results:

Example: 1

Customer	Sum(Sales)	Bottom(Sum(Sales))	Sum(Sales)+Bottom(Sum(Sales))	Bottom offset 3
	2566	757	3323	3105
Astrida	587	757	1344	1126
Betacab	539	757	1296	1078
Canutility	683	757	1440	1222
Divadip	757	757	1514	1296

The table visualization for Example 1.

In the screenshot of the table shown in this example, the table visualization is created from the dimension **Customer** and the measures: `sum(Sales)` and `Bottom(Sum(Sales))`.

The column **Bottom(Sum(Sales))** returns 757 for the all rows because this is the value of the bottom row: **Divadip**.

The table also shows more complex measures: one created from `sum(Sales)+Bottom(Sum(Sales))` and one labeled **Bottom offset 3**, which is created using the expression `sum(Sales)+Bottom(Sum(Sales), 3)` and has the argument **offset** set to 3. It adds the **Sum(Sales)** value for the current row to the value from the third row from the bottom row, that is, the current row plus the value for **Betacab**.

Example: 2

5 Functions in scripts and chart expressions

In the screenshots of tables shown in this example, more dimensions have been added to the visualizations: **Month** and **Product**. For charts with more than one dimension, the results of expressions containing the **Above**, **Below**, **Top**, and **Bottom** functions depend on the order in which the column dimensions are sorted by Qlik Sense. Qlik Sense evaluates the functions based on the column segments that result from the dimension that is sorted last. The column sort order is controlled in the properties panel under **Sorting** and is not necessarily the order in which the columns appear in a table.

In the first table, the expression is evaluated based on **Month**, and in the second table it is evaluated based on **Product**. The measure **End value** contains the expression `Bottom(Sum(Sales))`. The bottom row for **Month** is Dec, and the value for Dec both the values of **Product** shown in the screenshot is 22. (Some rows have been edited out of the screenshot to save space.)

Customer	Product	Month	Sum(Sales)	End value
			2566	-
Astrida	AA	Jan	46	22
Astrida	AA	Feb	60	22
Astrida	AA	Mar	70	22
Astrida	AA	Sep	78	22
Astrida	AA	Oct	12	22
Astrida	AA	Nov	78	22
Astrida	AA	Dec	22	22
Astrida	BB	Jan	46	22

First table for Example 2. The value of Bottom for the End value measure based on Month (Dec).

Customer	Product	Month	Sum(Sales)	End value
			2566	-
Astrida	AA	Jan	46	46
Astrida	BB	Jan	46	46
Astrida	AA	Feb	60	60
Astrida	BB	Feb	60	60
Astrida	AA	Mar	70	70
Astrida	BB	Mar	70	70
Astrida	AA	Apr	13	13
Astrida	BB	Apr	13	13

Second table for Example 2. The value of Bottom for the End value measure based on Product (BB for Astrida).

Please refer to Example: 2 in the **Above** function for further details.

Monthnames :

5 Functions in scripts and chart expressions

```
LOAD * INLINE [  
Month, Monthnumber  
Jan, 1  
Feb, 2  
Mar, 3  
Apr, 4  
May, 5  
Jun, 6  
Jul, 7  
Aug, 8  
Sep, 9  
Oct, 10  
Nov, 11  
Dec, 12  
];  
Sales2013:  
crosstable (Month, Sales) LOAD * inline [  
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec  
Astrida|46|60|70|13|78|20|45|65|78|12|78|22  
Betacab|65|56|22|79|12|56|45|24|32|78|55|15  
Canutility|77|68|34|91|24|68|57|36|44|90|67|27  
Divadip|57|36|44|90|67|27|57|68|47|90|80|94  
] (delimiter is '|');
```

To get the months to sort in the correct order, when you create your visualizations, go to the **Sorting** section of the properties panel, select **Month** and mark the checkbox **Sort by expression**. In the expression box write Monthnumber.

See also:

- [Top - chart function \(page 468\)](#)

Below - chart function

Below() evaluates an expression at a row below the current row within a column segment in a table. The row for which it is calculated depends on the value of **offset**, if present, the default being the row directly below. For charts other than tables, **Below()** evaluates for the row below the current column in the chart's straight table equivalent.

Syntax:

```
Below([TOTAL] expression [ , offset [,count ]])
```

Return data type: dual

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
offset	Specifying an offsetn , greater than 1 moves the evaluation of the expression n rows further

5 Functions in scripts and chart expressions

Argument	Description
	<p>down from the current row.</p> <p>Specifying an offset of 0 will evaluate the expression on the current row.</p> <p>Specifying a negative offset number makes the Below function work like the Above function with the corresponding positive offset number.</p>
count	<p>By specifying a third parameter count greater than 1, the function will return a range of count values, one for each of count table rows counting downwards from the original cell. In this form, the function can be used as an argument to any of the special range functions. <i>Range functions (page 476)</i></p>
TOTAL	<p>If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.</p>

On the last row of a column segment, a NULL value is returned, as there is no row below it.



A column segment is defined as a consecutive subset of cells having the same values for the dimensions in the current sort order. Inter-record chart functions are computed in the column segment excluding the right-most dimension in the equivalent straight table chart. If there is only one dimension in the chart, or if the TOTAL qualifier is specified, the expression evaluates across full table.



If the table or table equivalent has multiple vertical dimensions, the current column segment will include only rows with the same values as the current row in all dimension columns, except for the column showing the last dimension in the inter-field sort order.

Limitations:

Recursive calls will return NULL.

Examples and results:

Example 1:

Customer	Sum([Sales])	Below(Sum(Sales))	Sum(Sales)+Below(Sum(Sales))	Below + Offset 3	Higher
	2566	-	-	-	-
Astrida	587	539	1126	1344	Higher
Betacab	539	683	1222	-	-
Canutility	683	757	1440	-	-
Divadip	757	-	-	-	-

The table visualization for Example 1.

5 Functions in scripts and chart expressions

In the table shown in screenshot for Example 1, the table visualization is created from the dimension **Customer** and the measures: `Sum(Sales)` and `Below(Sum(Sales))`.

The column **Below(Sum(Sales))** returns NULL for the **Customer** row containing **Divadip**, because there is no row below it. The result for the row **Canutility** shows the value of `Sum(Sales)` for **Divadip**, the result for **Betacab** shows the value for `Sum(Sales)` for **Canutility**, and so on.

The table also shows more complex measures, which you can see in the columns labeled: `sum(Sales)+Below(Sum(Sales))`, **Below +Offset 3**, and **Higher?**. These expressions work as described in the following paragraphs.

For the column labeled **Sum(Sales)+Below(Sum(Sales))**, the row for **Astrida** shows the result of the addition of the `Sum(Sales)` values for the rows **Betacab + Astrida** (539+587). The result for the row **Betacab** shows the result of the addition of `Sum(Sales)` values for **Canutility + Betacab** (539+683).

The measure labeled **Below +Offset 3** created using the expression `sum(Sales)+Below(Sum(Sales), 3)` has the argument **offset**, set to 3, and has the effect of taking the value in the row three rows below the current row. It adds the `Sum(Sales)` value for the current **Customer** to the value from the **Customer** three rows below. The values for the lowest three **Customer** rows are null.

The measure labeled **Higher?** is created from the expression: `IF(Sum(Sales)>Below(Sum(Sales)), 'Higher')`. This compares the values of the current row in the measure `Sum(Sales)` with the row below it. If the current row is a greater value, the text "Higher" is output.



This function can also be used in charts other than tables, for example bar charts.



For other chart types, convert the chart to the straight table equivalent so you can easily interpret which row the function relates to.

For charts with more than one dimension, the results of expressions containing the **Above**, **Below**, **Top**, and **Bottom** functions depend on the order in which the column dimensions are sorted by Qlik Sense. Qlik Sense evaluates the functions based on the column segments that result from the dimension that is sorted last. The column sort order is controlled in the properties panel under **Sorting** and is not necessarily the order in which the columns appear in a table. Please refer to Example: 2 in the **Above** function for further details.

Example 2:	Result
The Below function can be used as input to the range functions. For example: <code>RangeAvg (Below(Sum(Sales),1,3))</code> .	Returns the average of the three results of the Sum(Sales) function evaluated on the three rows immediately below the current row.

Data used in examples:

```
Monthnames:
LOAD * INLINE [
Month, Monthnumber
Jan, 1
```

```
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];
Sales2013:
crosstable (Month, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

To get the months to sort in the correct order, when you create your visualizations, go to the **Sorting** section of the properties panel, select **Month** and mark the checkbox **Sort by expression**. In the expression box write Monthnumber.

See also:

- Above - chart function**
- Bottom - chart function (page 453)*
- Top - chart function (page 468)*
- RangeAvg (page 478)*

Column - chart function

Column() returns the value found in the column corresponding to **ColumnNo** in a straight table, disregarding dimensions. For example **Column(2)** returns the value of the second measure column.

Syntax:


```
Column (ColumnNo)
```

Return data type: dual

Arguments:

Argument	Description
ColumnNo	Column number of a column in the table containing a measure.

5 Functions in scripts and chart expressions

Argument	Description
	 <i>The Column() function disregards dimension columns.</i>

Limitations:

If **ColumnNo** references a column for which there is no measure, a NULL value is returned.

Recursive calls will return NULL.

Examples and results:

Example: Percentage total sales

Customer	Product	UnitPrice	UnitSales	Order Value	Total Sales Value	% Sales
A	AA	15	10	150	505	29.70
A	AA	16	4	64	505	12.67
A	BB	9	9	81	505	16.04
B	BB	10	5	50	505	9.90
B	CC	20	2	40	505	7.92
B	DD	25	-	0	505	0.00
C	AA	15	8	120	505	23.76
C	CC	19	-	0	505	0.00

Example: Percentage of sales for selected customer

Customer	Product	UnitPrice	UnitSales	Order Value	Total Sales Value	% Sales
A	AA	15	10	150	295	50.85
A	AA	16	4	64	295	21.69
A	BB	9	9	81	295	27.46

Examples	Results
Order Value is added to the table as a measure with the expression: <code>sum (UnitPrice*UnitSales)</code> .	The result of Column(1) is taken from the column Order Value, because this is the first measure column.

5 Functions in scripts and chart expressions

Examples	Results
<p>Total Sales Value is added as a measure with the expression: <code>Sum(TOTAL UnitPrice*UnitSales)</code></p> <p>% Sales is added as a measure with the expression <code>100*Column(1)/Column(2)</code></p>	<p>The result of Column(2) is taken from Total Sales Value, because this is the second measure column.</p> <p>See the results in the column % Sales in the example <i>Percentage total sales (page 460)</i>.</p>
<p>Make the selection Customer A.</p>	<p>The selection changes the Total Sales Value, and therefore the %Sales. See the example <i>Percentage of sales for selected customer (page 460)</i>.</p>

Data used in examples:

```
ProductData:
LOAD * inline [
Customer|Product|UnitSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD||25
Canutility|AA|8|15
Canutility|CC||19
] (delimiter is '|');
```

Dimensionality - chart function

Dimensionality() returns the number of dimensions for the current row.

Syntax:

```
Dimensionality ( )
```

Return data type:integer

Limitations:

This function is only available in charts. The number of dimensions in all rows, except the total which will be 0, will be returned.

Example:

A typical use for dimensionality is when you want to make a calculation only if there is a value present for a dimension.

Example	Result
<p>For a table containing the dimension UnitSales, you might only want to indicate an invoice is sent:</p> <pre>IF(Dimensionality()=3, "Invoiced").</pre>	

Exists

This script function determines whether a specific field value exists in a specified field of the data loaded so far. **Field** is a name or a string expression evaluating to a field name.

Syntax:

```
Exists(field [ , expression ] )
```

The field must exist in the data loaded so far by the script. **Expression** is an expression evaluating to the field value to look for in the specified field. If omitted, the current record's value in the specified field will be assumed.

Examples and results:

Example	Result
Exists(Month, 'Jan')	Returns -1 (True) if the field value 'Jan' is found in the current content of the field Month .
Exists(IDnr, IDnr)	Returns -1 (True) if the value of the field IDnr in the current record already exists in any previously read record containing that field.
Exists (IDnr)	Is identical to the previous example.
Load Employee, ID, Salary from Employees.csv; Load FirstName& '' &LastName as Employee, Comment from Citizens.csv where Exists (Employee, FirstName& '' &LastName);	Only comments regarding those citizens who are employees are read.
Load A, B, C, from Employees.csv where not Exists (A);	This is equivalent to performing a distinct load on field A.

FieldIndex

FieldIndex() returns the position of the field value **value** in the field **field_name** (by load order).

Syntax:

```
FieldIndex(field_name , value)
```

Return data type: integer

Arguments:

Argument	Description
field_name	Name of the field for which the index is required. Must be given as a string value. This means that the field name must be enclosed by single quotes.

5 Functions in scripts and chart expressions

Argument	Description
value	The value of the field field_name .

Limitations:

If **value** cannot be found among the field values of the field **field_name**, 0 is returned.

Examples and results:

The following example uses two fields: **First name** and **Initials**.

First name	Initials
John	JA
Sue	SB
Mark	MC
Peter	PD
Jane	JE
Peter	PF

Examples	Results
FieldIndex ('First name', 'John')	1, because 'John' appears first in the load order of the First name field. Note that in a filter pane John would appear as number 2 from the top as it's sorted alphabetically and not as in the load order.
FieldIndex ('First name', 'Peter')	4, because FieldIndex() returns only one value, that is the first occurrence in the load order.

Data used in example:

```
Initials:
LOAD * inline [
"First name"|Initials|"Has cellphone"
John|JA|Yes
Sue|SB|Yes
Mark|MC |No
Peter|PD|No
Jane|JE|Yes
Peter|PF|Yes ] (delimiter is '|');
```

FieldValue

FieldValue() returns the value found in position **elem_no** of the field **field_name** (by load order).

5 Functions in scripts and chart expressions

Syntax:

```
FieldValue (field_name , elem_no)
```

Return data type: dual

Arguments:

Argument	Description
field_name	Name of the field for which the value is required. Must be given as a string value. This means that the field name must be enclosed by single quotes.
elem_no	The position (element) number of the field, following the load order, that the value is returned for.

Limitations:

If **elem_no** is larger than the number of field values, NULL is returned.

First name	Initials
John	JA
Sue	SB
Mark	MC
Peter	PD
Jane	JE
Peter	PF

Examples and results:

The following example uses two fields: **First name** and **Initials**.

Examples	Results
FieldValue ('First name', '1')	John as John appears first in the load order of the First name field. Note that in a filter pane John would appear as number 2 from the top, after Jane , as it's sorted alphabetically and not as in the load order.
FieldValue ('First name', '7')	NULL as there are only 6 values in the First name field.

Data used in example:

Initials:

5 Functions in scripts and chart expressions

```
LOAD * inline [  
"First name"|Initials|"Has cellphone"  
John|JA|Yes  
Sue|SB|Yes  
Mark|MC |No  
Peter|PD|No  
Jane|JE|Yes  
Peter|PF|Yes ] (delimiter is '|');
```

FieldValueCount

FieldValueCount() is an **integer** function that finds the number of distinct values in a field.

Syntax:

```
FieldValueCount (field_name)
```

Arguments:

Argument	Description
field_name	Name of the field for which the value is required. Must be given as a string value. This means that the field name must be enclosed by single quotes.

Examples and results:

The following example uses two fields: **First name** and **Initials**.

Examples	Results
FieldValueCount('First name')	5 as Peter appears twice.
FieldValueCount('Initials')	6 as Initials only has distinct values.

Data used in example:

First name	Initials
John	JA
Sue	SB
Mark	MC
Peter	PD
Jane	JE
Peter	PF

LookUp

This script function returns the value of **fieldname** corresponding to the first occurrence of the value **matchfieldvalue** in the field **matchfieldname**.

Syntax:

```
lookup(fieldname, matchfieldname, matchfieldvalue [, tablename])
```

The search order is load order unless the table is the result of complex operations such as joins, in which case the order is not well defined. Both **fieldname** and **matchfieldname** must be fields in the same table, specified by **tablename**.

If no match is found, NULL is returned.

Arguments:

Argument	Description
fieldname	Name of field to return value. Input value must be given as a string (for example quoted literals).
matchfieldname	Name of field to lookup matchfieldvalue in. Input value must be given as a string (for example quoted literals).
matchfieldvalue	Value to lookup in matchfieldname field
tablename	Name of table. Input value must be given as a string (for example quoted literals). If tablename is omitted the current table is assumed.

Example:

```
LookUp('Price', 'ProductID', InvoicedProd, 'pricelist')
```

NoOfRows - chart function

NoOfRows() returns the number of rows in the current column segment in a table. For bitmap charts, **NoOfRows()** returns the number of rows in the chart's straight table equivalent.

If the table or table equivalent has multiple vertical dimensions, the current column segment will include only rows with the same values as the current row in all dimension columns, except for the column showing the last dimension in the inter-field sort order.

Syntax:

```
NoOfRows ( [TOTAL] )
```

Return data type: integer

5 Functions in scripts and chart expressions

Arguments:

Argument	Description
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.

Example:

```
if( RowNo( )= NoOfRows( ), 0, Above( sum( Sales )))
```

See also:

▫ [RowNo - chart function \(page 340\)](#)

Peek

This script function returns the contents of the **fieldname** in the record specified by **row** in the internal table **tablename**. Data are fetched from the associative Qlik Sense database.

Syntax:

```
Peek(fieldname [ , row [ , tablename ] ] )
```

Arguments:

Argument	Description
Fieldname	Must be given as a string (e.g. a quoted literal).
Row	Must be an integer. 0 denotes the first record, 1 the second and so on. Negative numbers indicate order from the end of the table. -1 denotes the last record read. If no row is stated, -1 is assumed.
Tablename	A table label without the ending colon. If no tablename is stated, the current table is assumed. If used outside the LOAD statement or referring to another table, the tablename must be included.

Examples and results:

Example	Result
Peek('Sales')	Returns the value of Sales in the previous record read (equivalent to <code>Previous(Sales)</code>).

5 Functions in scripts and chart expressions

Example	Result
Peek('Sales', 2)	Returns the value of Sales from the third record read from the current internal table.
Peek('Sales', -2)	Returns the value of Sales from the second last record read into the current internal table.
Peek('Sales', 0, 'Tab1')	Returns the value of Sales from the first record read into the input table labeled Tab1.
LOAD A, B, numsum(B, Peek('Bsum')) as Bsum...;	Creates an accumulation of B in Bsum .

Previous

This script function returns the value of **expression** using data from the previous input record that was not discarded due to a **where** clause. In the first record of an internal table the function will return NULL.

Syntax:

```
Previous( expression )
```

The **previous** function may be nested in order to access records further back. Data are fetched directly from the input source, making it possible to refer also to fields which have not been loaded into Qlik Sense, i.e. even if they have not been stored in its associative database.

Examples:

```
LOAD *, Sales / Previous(Sales) as Increase from ...;  
LOAD A, Previous(Previous( A )) as B from ...;
```

Top - chart function

Top() evaluates an expression at the first (top) row of a column segment in a table. The row for which it is calculated depends on the value of **offset**, if present, the default being the top row. For charts other than tables, the **Top()** evaluation is made on the first row of the current column in the chart's straight table equivalent.

Syntax:

```
Top( [[TOTAL] expr [ , offset [, count ] ] )
```

Return data type: dual

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
expr	The expression or field containing the data to be measured.
offset	<p>Specifying an offset of n, greater than 1, moves the evaluation of the expression down n rows below the top row.</p> <p>Specifying a negative offset number makes the Top function work like the Bottom function with the corresponding positive offset number.</p>
count	By specifying a third parameter count greater than 1, the function will return a range of count values, one for each of the last count rows of the current column segment. In this form, the function can be used as an argument to any of the special range functions. <i>Range functions (page 476)</i>
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.



A column segment is defined as a consecutive subset of cells having the same values for the dimensions in the current sort order. Inter-record chart functions are computed in the column segment excluding the right-most dimension in the equivalent straight table chart. If there is only one dimension in the chart, or if the **TOTAL** qualifier is specified, the expression evaluates across full table.



If the table or table equivalent has multiple vertical dimensions, the current column segment will include only rows with the same values as the current row in all dimension columns, except for the column showing the last dimension in the inter-field sort order.

Limitations:

Recursive calls will return NULL.

Examples and results:

Example: 1

Top and Bottom					
Customer	Q	Sum(Sales)	Top(Sum(Sales))	Sum(Sales)+Top(Sum(Sales))	Top offset 3
Totals		2566	587	3153	3249
Astrida		587	587	1174	1270
Betacab		539	587	1126	1222
Canutility		683	587	1270	1366
Divadip		757	587	1344	1440

5 Functions in scripts and chart expressions

In the screenshot of the table shown in this example, the table visualization is created from the dimension **Customer** and the measures: `Sum(Sales)` and `Top(Sum(Sales))`.

The column **Top(Sum(Sales))** returns 587 for the all rows because this is the value of the top row: **Astrida**.

The table also shows more complex measures: one created from `Sum(Sales)+Top(Sum(Sales))` and one labeled **Top offset 3**, which is created using the expression `Sum(Sales)+Top(Sum(Sales), 3)` and has the argument **offset** set to 3. It adds the **Sum(Sales)** value for the current row to the value from the third row from the top row, that is, the current row plus the value for **Canutility**.

Example: 2

In the screenshots of tables shown in this example, more dimensions have been added to the visualizations: **Month** and **Product**. For charts with more than one dimension, the results of expressions containing the **Above**, **Below**, **Top**, and **Bottom** functions depend on the order in which the column dimensions are sorted by Qlik Sense. Qlik Sense evaluates the functions based on the column segments that result from the dimension that is sorted last. The column sort order is controlled in the properties panel under **Sorting** and is not necessarily the order in which the columns appear in a table.

Customer	Product	Month	Sum(Sales)	First value
			2566	-
Astrida	AA	Jan	46	46
Astrida	AA	Feb	60	46
Astrida	AA	Mar	70	46
Astrida	AA	Apr	13	46
Astrida	AA	May	78	46
Astrida	AA	Jun	20	46
Astrida	AA	Jul	45	46
Astrida	AA	Aug	65	46
Astrida	AA	Sep	78	46
Astrida	AA	Oct	12	46
Astrida	AA	Nov	78	46
Astrida	AA	Dec	22	46

First table for Example 2. The value of Top for the First value measure based on Month (Jan).

5 Functions in scripts and chart expressions

Customer	Product	Month	Sum(Sales)	First value
			2566	-
Astrida	AA	Jan	46	46
Astrida	BB	Jan	46	46
Astrida	AA	Feb	60	60
Astrida	BB	Feb	60	60
Astrida	AA	Mar	70	70
Astrida	BB	Mar	70	70
Astrida	AA	Apr	13	13
Astrida	BB	Apr	13	13

Second table for Example 2. The value of Top for the First value measure based on Product (AA for Astrida).

Please refer to Example: 2 in the **Above** function for further details.

Monthnames:

```
LOAD * INLINE [  
Month, Monthnumber  
Jan, 1  
Feb, 2  
Mar, 3  
Apr, 4  
May, 5  
Jun, 6  
Jul, 7  
Aug, 8  
Sep, 9  
Oct, 10  
Nov, 11  
Dec, 12  
];  
Sales2013:
```

```
crosstable (Month, Sales) LOAD * inline [  
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec  
Astrida|46|60|70|13|78|20|45|65|78|12|78|22  
Betacab|65|56|22|79|12|56|45|24|32|78|55|15  
Canutility|77|68|34|91|24|68|57|36|44|90|67|27  
Divadip|57|36|44|90|67|27|57|68|47|90|80|94  
] (delimiter is '|');
```

To get the months to sort in the correct order, when you create your visualizations, go to the **Sorting** section of the properties panel, select **Month** and mark the checkbox **Sort by expression**. In the expression box write Monthnumber.

See also:

- *Bottom - chart function (page 453)*
- *Above - chart function (page 449)*

- ▢ *Sum - chart function (page 158)*
- ▢ *RangeAvg (page 478)*
- ▢ *Range functions (page 476)*

5.15 Logical functions

This section describes functions handling logical operations. All functions can be used in both the data load script and in chart expressions.

IsNum

Returns -1 (True) if the expression can be interpreted as a number, otherwise 0 (False).

```
IsNum( expr )
```

IsText

Returns -1 (True) if the expression has a text representation, otherwise 0 (False).

```
IsText( expr )
```

5.16 Mapping functions

This section describes functions for handling mapping tables. A mapping table can be used to replace field values or field names during script execution.

Mapping functions can only be used in the data load script.

Mapping functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

ApplyMap

This script function is used for mapping any expression to a previously loaded mapping table.

```
ApplyMap ('mapname', expr [ , defaultexpr ] )
```

MapSubstring

This script function is used to map parts of any expression to a previously loaded mapping table. The mapping is case sensitive and non-iterative and substrings are mapped from left to right.

```
MapSubstring ('mapname', expr)
```

ApplyMap

This script function is used for mapping any expression to a previously loaded mapping table.

Syntax:

5 Functions in scripts and chart expressions

```
ApplyMap('mapname', expr [ , defaultexpr ] )
```

Arguments:

Argument	Description
mapname	The name of a mapping table that has previously been created through the mapping load or the mapping select statement. Its name must be enclosed by single, straight quotation marks.
expr	The expression, the result of which should be mapped.
defaultexpr	An optional expression which will be used as a default mapping value if the mapping table does not contain a matching value for expr . If no default value is given, the value of expr will be returned as is.

Example:

```
// Assume the following mapping table:  
map1:  
mapping LOAD * inline [  
x, y  
1, one  
2, two  
3, three ] ;
```

```
ApplyMap ('map1', 2 ) returns 'two'
```

```
ApplyMap ('map1', 4 ) returns 4
```

```
ApplyMap ('map1', 5, 'xxx') returns 'xxx'
```

```
ApplyMap ('map1', 1, 'xxx') returns 'one'
```

```
ApplyMap ('map1', 5, null( ) ) returns NULL
```

```
ApplyMap ('map1', 3, null( ) ) returns 'three'
```

MapSubstring

This script function is used to map parts of any expression to a previously loaded mapping table. The mapping is case sensitive and non-iterative and substrings are mapped from left to right.

Syntax:

```
MapSubstring('mapname', expr)
```

This function can be used for mapping parts of any expression on a previously loaded mapping table. The mapping is case sensitive and non-recursive. The substrings are mapped from the left to the right.

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
mapname	The name of a mapping table previously read by a mapping load or a mapping select statement. The name must be enclosed by single straight quotation marks. Expr is the expression whose result should be mapped by substrings.

Example:

```
// Assume the following mapping table:
map1:
mapping LOAD * inline [
x, y
1, <one>
aa, XYZ
x, b ] ;
```

```
MapSubstring ('map1', 'A123') returns 'A<one>23'
```

```
MapSubstring ('map1', 'baaar') returns 'bXYZar'
```

```
MapSubstring ('map1', 'xaa1') returns 'bXYZ<one>'
```

5.17 Mathematical constants and parameter-free functions

This section describes functions for mathematical constants and boolean values. These functions do not have any parameters, but the parentheses are still required.

All functions can be used in both the data load script and in chart expressions.

e

The function returns the base of the natural logarithms, **e**. (2.71828...)

```
e ( )
```

false

Returns a dual value with text value 'false' and numeric value 0, which can be used as logical false in expressions.

```
false ( )
```

pi

The function returns the value of π (3.14159...)

```
pi ( )
```

rand

Returns a random number between 0 and 1.

```
rand ( )
```

true

5 Functions in scripts and chart expressions

Returns a dual value with text value 'true' and numeric value -1, which can be used as logical true in expressions.

```
true( )
```

5.18 NULL functions

This section describes functions for returning or detecting NULL values.

All functions can be used in both the data load script and in chart expressions.

NULL functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Null

The **Null** function returns a NULL value.

```
NULL ( )
```

IsNull

The **IsNull** function tests if the parameter value is NULL and if so, returns -1 (True), otherwise 0 (False). Note that a string with length zero is not considered as a NULL and will cause **IsNull** to return False.

```
IsNull (expr )
```

IsNull

The **IsNull** function tests if the parameter value is NULL and if so, returns -1 (True), otherwise 0 (False). Note that a string with length zero is not considered as a NULL and will cause **IsNull** to return False.

Syntax:

```
IsNull (expr )
```

Example:

```
If(IsNull( x ), 0, x )
```

NULL

The **Null** function returns a NULL value.

Syntax:

```
Null ( )
```

Example:

If(len(trim(x))= 0 or x='NULL', Null(), x)

5.19 Range functions

All range functions can be used in both the data load script and in chart expressions.



*Range functions replace the following general numeric functions: **numsum**, **numavg**, **numcount**, **nummin** and **nummax**, which should now be regarded as obsolete.*

Basic range functions

RangeMax

RangeMax() returns the highest numeric values found within the expression or field.

```
RangeMax (first_expr {, Expression})
```

RangeMaxString

RangeMaxString() returns the last value in the text sort order that it finds in the expression or field.

```
RangeMaxString (first_expr {, Expression})
```

RangeMin

Min() returns the lowest numeric values found within the expression or field.

```
RangeMin (first_expr {, Expression})
```

RangeMinString

RangeMinString() returns the first value in the text sort order that it finds in the expression or field.

```
RangeMinString (first_expr {, Expression})
```

RangeMode

RangeMode() finds the most commonly occurring value (mode value) in the expression or field.

```
RangeMode (first_expr {, Expression})
```

RangeOnly

RangeOnly() is a **dual** function that returns a value if the expression evaluates to one unique value. If this is not the case then **NULL** is returned.

```
RangeOnly (first_expr {, Expression})
```

RangeSum

RangeSum() returns the sum of a range of values. All non-numeric values are treated as 0, unlike the **+** operator.

```
RangeSum (first_expr {, Expression})
```

Counter range functions

RangeCount

RangeCount() returns the number of values, text and numeric, found within the specified range or expression.

```
RangeCount(first_expr {, Expression})
```

RangeMissingCount

RangeMissingCount() finds the number of non-numeric values (including NULL) in the expression or field.

```
RangeMissingCount(first_expr {, Expression})
```

RangeNullCount

RangeNullCount() finds the number of NULL values in the expression or field.

```
RangeNullCount(first_expr {, Expression})
```

RangeNumericCount

RangeNumericCount() finds the number of numeric values in an expression or field.

```
RangeNumericCount(first_expr {, Expression})
```

RangeTextCount

RangeTextCount() returns the number of text values in an expression or field.

```
RangeTextCount(first_expr {, Expression})
```

Statistical range functions

RangeAvg

RangeAvg() returns the average of a range. Input to the function can be either a range of values or an expression.

```
RangeAvg(first_expr {, Expression})
```

RangeCorrel

RangeCorrel() returns the correlation coefficient for two sets of data. The correlation coefficient is a measure of the relationship between the data sets.

```
RangeCorrel(x_values , y_values {,Expression})
```

RangeFractile

RangeFractile() returns the value that corresponds to the n-th **fractile** (quantile) of a range of numbers.

```
RangeFractile(fractile, first_expr { ,Expression})
```

RangeKurtosis

RangeKurtosis() returns the value that corresponds to the kurtosis of a range of numbers.

5 Functions in scripts and chart expressions

```
RangeKurtosis(expr1 [ , expr2, ... exprN ])
```

RangeSkew

RangeSkew() returns the value corresponding to the skewness of a range of numbers.

```
RangeSkew(expr1 [ , expr2, ... exprN ])
```

RangeStdev

RangeStdev() finds the standard deviation of a range of numbers.

```
RangeStdev(expr1 [ , expr2, ... exprN ])
```

See also:

▫ [Inter-record functions \(page 446\)](#)

RangeAvg

RangeAvg() returns the average of a range. Input to the function can be either a range of values or an expression.

Syntax:

```
RangeAvg (first_expr {, Expression})
```

Return data type: numeric

Arguments:

The argument expressions of this function may contain inter-record functions with a third optional parameter, which in themselves return a range of values.

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

If no numeric value is found, NULL is returned.

Examples and results:

Examples	Results
RangeAvg (1,2,4)	Returns 2.33333333

5 Functions in scripts and chart expressions

Examples	Results
RangeAvg (1, 'xyz')	Returns 1
RangeAvg (null(), 'abc')	Returns NULL

Example with expression:

```
RangeAvg (MyField, Above(MyField), Above(Above(MyField)))
```

Returns a sliding average of the result of the range of three values of **MyField** calculated on the current row and two rows above the current row.

Data used in examples:

MyField	RangeAvg (Above(Above(MyField), Above(Above(MyField))))
10	10
2	6
8	6.666666667
18	9.333333333
5	10.33333333
9	10.66666667

```
RangeTab:
LOAD * INLINE [
MyField
10
2
8
18
5
9
] (delimiter is '|');
```

See also:

- ▢ [Avg - chart function \(page 190\)](#)
- ▢ [Count - chart function \(page 163\)](#)

RangeCorrel

RangeCorrel() returns the correlation coefficient for two sets of data. The correlation coefficient is a measure of the relationship between the data sets.

Syntax:

5 Functions in scripts and chart expressions

RangeCorrel (x_values , y_values {, Expression})

Return data type: numeric

If you provide values manually, enter them as (x,y) pairs. For example, to evaluate two series of data, array 1 and array 2, where the array 1 = 2,6,9 and array 2 = 3,8,4 you would write `RangeCorrel (2,3,6,8,9,4)` which returns 0.269.

Arguments:

Argument	Description
x-value, y-value	Each value represents a single value or a range of values as returned by an inter-record functions with a third optional parameter. Each value or range of values must correspond to an x-value or a range of y-values .
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

The function needs at least two pairs of coordinates to be calculated.

Text values, NULL values and missing values are disregarded.

Examples and results:

Examples	Results
<code>RangeCorrel (2,3,6,8,9,4)</code>	Returns 0.269

See also:

▢ [Correl - chart function \(page 192\)](#)

RangeCount

RangeCount() returns the number of values, text and numeric, found within the specified range or expression.

Syntax:

RangeCount (first_expr {,Expression})

Return data type: integer

Arguments:

The argument expressions of this function may contain inter-record functions with a third optional parameter, which in themselves return a range of values.

5 Functions in scripts and chart expressions

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

NULL values are not counted.

Examples and results:

Examples	Results
RangeCount (1,2,4)	Returns 3
RangeCount (2, 'xyz')	Returns 2
RangeCount (null())	Returns 0
RangeCount (2, 'xyz', null())	Returns 2

Example with expression:

```
RangeCount (Above(Sum(MyField),1,3))
```

Returns the number of values within the three results of the **Sum(MyField)** function above the current row.

Data used in examples:

MyField	RangeCount(Above(Sum(MyField),1,3))
23	0
63	1
74	2
89	3
44	3
54	3

See also:

▢ [Count - chart function \(page 163\)](#)

RangeFractile

RangeFractile() returns the value that corresponds to the n-th **fractile** (quantile) of a range of numbers.

5 Functions in scripts and chart expressions

Syntax:

```
RangeFractile(fractile, first_expr { ,Expression})
```

Return data type: numeric

Arguments:

The argument expressions of this function may contain inter-record functions with a third optional parameter, which in themselves return a range of values.

Argument	Description
fractile	A number between 0 and 1 corresponding to the fractile (quantile expressed as a fraction) to be calculated.
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Examples and results:

Examples	Results
RangeFractile (0.24,1,2,4,6)	Returns 1.72
RangeFractile(0.5,1,2,3,4,6)	Returns 3
RangeFractile (0.5,1,2,5,6)	Returns 3.5

Example with expression:

```
RangeFractile (0.5, Above(Sum(MyField),1,3))
```

Returns the fractile from the range of values within the three results of the **Sum(MyField)** function above the current row.

Data used in examples:

MyField	RangeFractile(0.5, Above(Sum(MyField),1,3))
1	-
2	1
3	2
4	3
6	3

See also:

▢ [Fractile - chart function \(page 194\)](#)

RangeIRR

This script function returns the internal rate of return for a series of cash flows represented by the numbers in values. These cash flows do not have to be even, as they would be for an annuity. However, the cash flows must occur at regular intervals, such as monthly or annually. The internal rate of return is the interest rate received for an investment consisting of payments (negative values) and income (positive values) that occur at regular periods.

Syntax:

```
RangeIRR( value { ,value} )
```

Arguments:

Argument	Description
Value	A single value or a range of values as returned by an inter record function with a third optional parameter. The function needs at least one positive and one negative value to be calculated. Text values, NULL values and missing values are disregarded.

Examples:

RangeIRR(-70000,12000,15000,18000,21000,26000) returns 0,0866

```
RangeIRR(above(sum(value), 0, 10))
```

```
RangeIRR(above(total value, 0, rowno(total)))
```

See also:

▢ [Inter-record functions \(page 446\)](#)

RangeKurtosis

RangeKurtosis() returns the value that corresponds to the kurtosis of a range of numbers.

Syntax:

```
RangeKurtosis (first_expr { ,Expression})
```

Return data type: numeric

Arguments:

5 Functions in scripts and chart expressions

The argument expressions of this function may contain inter-record functions with a third optional parameter, which in themselves return a range of values.

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

If no numeric value is found, NULL is returned.

Examples and results:

Examples	Results
RangeKurtosis (1,2,4,7)	Returns -0.28571428571429
RangeKurtosis (Above (Count(MyField),0,3))	Returns a sliding kurtosis of the result of the inner count(MyField) expression calculated on the current row and two rows above the current row.

Data used in examples:

```
RangeTab:
LOAD * INLINE [
MyField
10
2
8
18
5
9
] (delimiter is '|');
```

See also:

▢ [Kurtosis - chart function \(page 197\)](#)

RangeMax

RangeMax() returns the highest numeric values found within the expression or field.

Syntax:

```
RangeMax (first_expr { , Expression})
```

Return data type: numeric

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

If no numeric value is found, NULL is returned.

Examples and results:

Examples	Results
RangeMax (1,2,4)	Returns 4
RangeMax (1,'xyz')	Returns 1
RangeMax (null(), 'abc')	Returns NULL

Example with expression:

```
RangeMax (Above(Sum(MyField),1,3))
```

Returns the highest of the three results of the **Sum(MyField)** function above the current row.

The first row will return NULL because there is no row above to aggregate on.

Data used in examples:



*Disable sorting of **MyField** to ensure that example works as expected.*

MyField	RangeMax (Above(Sum(MyField),1,3))
10	-
2	10
8	10
18	10
5	18
9	18

Data used in examples:

```
RangeTab:  
LOAD * INLINE [  
MyField  
10
```

```
2
8
18
5
9
] (delimiter is '|');
```

RangeMaxString

RangeMaxString() returns the last value in the text sort order that it finds in the expression or field.

Syntax:

```
RangeMaxString(first_expr { , Expression})
```

Return data type: string

Arguments:

The argument expressions of this function may contain inter-record functions with a third optional parameter, which in themselves return a range of values.

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Examples and results:

Examples	Results
RangeMaxString (1,2,4)	Returns 4
RangeMaxString ('xyz', 'abc')	Returns 'xyz'
RangeMaxString (5, 'abc')	Returns 'abc'
RangeMaxString (null())	Returns NULL

Example with expression:

```
RangeMaxString (Above(MaxString(MyField),0,3))
```

Returns the last (in text sort order) of the three results of the **MaxString(MyField)** function evaluated on the current row and two rows above the current row.

Data used in examples:



*Disable sorting of **MyField** to ensure that example works as expected.*

MyField	RangeMaxString(Above(MaxString(MyField),0,3))
10	10
abc	abc
8	abc
def	def
xyz	xyz
9	xyz

See also:

- [MaxString - chart function \(page 322\)](#)

RangeMin

Min() returns the lowest numeric values found within the expression or field.

Syntax:

```
RangeMin (first_expr {,Expression})
```

Return data type: numeric

Arguments:

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

If no numeric value is found, NULL is returned.

Examples and results:

Examples	Results
RangeMin (1,2,4)	Returns 1
RangeMin (1,'xyz')	Returns 1
RangeMin (null(), 'abc')	Returns NULL

Example with expression:

5 Functions in scripts and chart expressions

`RangeMin (Above(Sum(MyField),0,3))`

Returns the lowest of the three results of the **Sum(MyField)** function evaluated on the current row and the two rows above the current row.

MyField	RangeMin(Above(Sum(MyField),0,3))
10	10
2	2
8	2
18	2
5	5
9	5

Data used in examples:

```
RangeTab:
LOAD * INLINE [
MyField
10
2
8
18
5
9
] (delimiter is '|');
```

See also:

- ▢ [Min - chart function \(page 151\)](#)

RangeMinString

RangeMinString() returns the first value in the text sort order that it finds in the expression or field.

Syntax:

```
RangeMinString(first_expr {, Expression})
```

Return data type: string

Arguments:

The argument expressions of this function may contain inter-record functions with a third optional parameter, which in themselves return a range of values.

5 Functions in scripts and chart expressions

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Examples and results:

Examples	Results
<code>RangeMinString (1,2,4)</code>	Returns 1
<code>RangeMinString ('xyz', 'abc')</code>	Returns 'abc'
<code>RangeMinString (5, 'abc')</code>	Returns 5
<code>RangeMinString (null())</code>	Returns NULL

Example with expression:

```
RangeMinString (Above(MinString(MyField),0,3))
```

Returns the first (in text sort order) of the three results of the **MinString(MyField)** function evaluated on the current row and two rows above the current row.

Data used in examples:



*Disable sorting of **MyField** to ensure that example works as expected.*

MyField	RangeMinString(Above(MinString(MyField),0,3))
10	10
abc	10
8	8
def	8
xyz	8
9	9

See also:

- ▢ [MinString - chart function \(page 324\)](#)

RangeMissingCount

RangeMissingCount() finds the number of non-numeric values (including NULL) in the expression or field.

5 Functions in scripts and chart expressions

Syntax:

```
RangeMissingCount (first_expr {, Expression})
```

Return data type: integer

Arguments:

The argument expressions of this function may contain inter-record functions with a third optional parameter, which in themselves return a range of values.

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Examples and results:

Examples	Results
RangeMissingCount (1,2,4)	Returns 0
RangeMissingCount (5, 'abc')	Returns 1
RangeMissingCount (null())	Returns 1

Example with expression:

```
RangeMissingCount (Above(MinString(MyField),0,3))
```

Returns the number of non-numeric values in the three results of the **MinString(MyField)** function evaluated on the current row and two rows above the current row.

Data used in examples:



*Disable sorting of **MyField** to ensure that example works as expected.*

MyField	RangeMissingCount(Above(MinString(MyField),0,3))
10	Returns 2 because there are no rows above this row so 2 of the 3 values are missing.
abc	Returns 2 because there is only 1 row above the current row and the current row is non-numeric ('abc').
8	Returns 1 because 1 of the 3 rows includes a non-numeric ('abc').
def	Returns 2 because 2 of the 3 rows include non-numeric values ('def' and 'abc').

5 Functions in scripts and chart expressions

MyField `RangeMissingCount(Above(MinString(MyField),0,3))`

xyz Returns 2 because 2 of the 3 rows include non-numeric values (' xyz' and 'def').

9 Returns 2 because 2 of the 3 rows include non-numeric values (' xyz' and 'def').

See also:

▫ [MissingCount - chart function \(page 166\)](#)

RangeMode

RangeMode() finds the most commonly occurring value (mode value) in the expression or field.

Syntax:

```
RangeMode (first_expr {, Expression})
```

Return data type: numeric

Arguments:

The argument expressions of this function may contain inter-record functions with a third optional parameter, which in themselves return a range of values.

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

If more than one value shares the highest frequency, NULL is returned.

Examples and results:

Examples	Results
<code>RangeMode (1,2,9,2,4)</code>	Returns 2
<code>RangeMode ('a',4, 'a',4)</code>	Returns NULL
<code>RangeMode (null())</code>	Returns NULL

Example with expression:

```
RangeMode (Above(Sum(Temperature),0,3))
```

5 Functions in scripts and chart expressions

Returns the most commonly occurring value in the three results of the **Sum(Temperature)** function evaluated on the current row and two rows above the current row.

Data used in example:



*Disable sorting of **MyField** to ensure that example works as expected.*

Region	City	Temperature	RangeMode(Above(Sum(Temperature),0,3))
A	A	18	18
A	B	17	-
A	C	16	-
A	D	18	-
A	E	16	16
A	F	20	-
A	G	22	-
A	H	20	20

Example with expression on a single field with Sum():

RangeMode (Above(Sum(MyField),0,3))

Returns the most commonly occurring value in the three results of the **Sum(MyField)** function evaluated on the current row and two rows above the current row.

Data used in example:



*Disable sorting of **MyField** to ensure that example works as expected.*

MyField	Sum (MyField)	RangeMode(Above(Sum(MyField),0,3))
10	10	Returns 10 because there are no rows above so the single value is the most commonly occurring.
2	2	
8	8	
18	18	

5 Functions in scripts and chart expressions

MyField	Sum (MyField)	RangeMode(Above(Sum(MyField),0,3))
5	5	
9	18	Returns 18 because RangeMode is evaluating Sum(MyField) where 18 occurs twice in 3 rows.
7	7	
9		



Load order should be maintained in the straight table.

```
angeTemp:
LOAD * INLINE [
Region|City|Temperature
A|A|18
A|B|17
A|C|16
A|D|18
A|E|16
A|F|20
A|G|22
A|H|20
] (delimiter is '|');
```

See also:

▢ [Mode - chart function \(page 154\)](#)

RangeNPV

This script function returns the net present value of an investment based on a discount **rate** and a series of future payments (negative values) and incomes (positive values). The result has a default number format of **money**.

Syntax:

```
RangeNPV ( rate, value { ,value} )
```

Arguments:

Argument	Description
rate	The interest rate per period

5 Functions in scripts and chart expressions

Argument	Description
value	A payment or income occurring at the end of each period. Each value may be a single value or a range of values as returned by an inter-record function with a third optional parameter. Text values, NULL values and missing values are disregarded.

Examples:

`RangeNPV(0.1, -10000, 3000, 4200, 6800)` returns 1188,44

`RangeNPV(0.05, above(sum(value), 0, 10))`

`RangeNPV(0.05, above(total value, 0, rowno(total)))`

See also:

▢ [Inter-record functions \(page 446\)](#)

RangeNullCount

RangeNullCount() finds the number of NULL values in the expression or field.

Syntax:

```
RangeNullCount (firstexpr [,Expression])
```

Return data type: integer

Arguments:

The argument expressions of this function may contain inter-record functions with a third optional parameter, which in themselves return a range of values.

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Examples and results:

Examples	Results
<code>RangeNullCount (1,2,4)</code>	Returns 0
<code>RangeNullCount (5, 'abc')</code>	Returns 0
<code>RangeNullCount (null(), null())</code>	Returns 2

Example with expression:

5 Functions in scripts and chart expressions

`RangeNullCount (Above(Sum(MyField),0,3))`

Returns the number of NULL values in the three results of the **Sum(MyField)** function evaluated on the current row and two rows above the current row.

Data used in examples:



*Copying **MyField** in example below will not result in NULL value.*

MyField	RangeNullCount(Above(Sum(MyField),0,3))
10	Returns 2 because there are no rows above this row so 2 of the 3 values are missing (=NULL).
2	Returns 1 because there is only 1 row above the current row so 1 of the 3 values is missing (=NULL).
8	Returns 1 because 1 of the 3 rows includes a non-numeric ('abc').
null	Returns 1 because current row is a NULL value.
5	Returns 1 because row above is a NULL value.
9	Returns 1 because value 2 rows above the current row is a NULL value.

See also:

- ▢ [NullCount - chart function \(page 168\)](#)

RangeNumericCount

RangeNumericCount() finds the number of numeric values in an expression or field.

Syntax:

```
RangeNumericCount (first_expr {, Expression})
```

Return data type: integer

Arguments:

The argument expressions of this function may contain inter-record functions with a third optional parameter, which in themselves return a range of values.

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Examples and results:

Examples	Results
<code>RangeNumericCount (1,2,4)</code>	Returns 3
<code>RangeNumericCount (5, 'abc')</code>	Returns 1
<code>RangeNumericCount (null())</code>	Returns 0

Example with expression:

```
RangeNumericCount (Above(MaxString(MyField),0,3))
```

Returns the number of numeric values in the three results of the **MaxString(MyField)** function evaluated on the current row and two rows above the current row.

Data used in examples:



*Disable sorting of **MyField** to ensure that example works as expected.*

MyField	RangeNumericCount(Above(MaxString(MyField),0,3))
10	1
abc	1
8	2
def	1
xyz	1
9	1

See also:

▢ [NumericCount - chart function \(page 170\)](#)

RangeOnly

RangeOnly() is a **dual** function that returns a value if the expression evaluates to one unique value. If this is not the case then **NULL** is returned.

Syntax:

```
RangeOnly (first_expr {, Expression})
```

Return data type: dual

5 Functions in scripts and chart expressions

Arguments:

The argument expressions of this function may contain inter-record functions with a third optional parameter, which in themselves return a range of values.

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Examples and results:

Examples	Results
<code>RangeOnly (1,2,4)</code>	Returns NULL
<code>RangeOnly (5, 'abc')</code>	Returns NULL
<code>RangeOnly (null(), 'abc')</code>	Returns 'abc'
<code>RangeOnly(10,10,10)</code>	Returns 10

Example with expression:

```
RangeOnly (Above(Sum(MyField),0,3))
```

Returns one value if the three results of the **Sum(MyField)** function over the current row and two rows above the current row contain exactly one value.

Data used in examples:



*Disable sorting of **MyField** to ensure that example works as expected.*

MyField	Sum (MyField)	RangeOnly(Above(Sum(MyField),0,3))
10	10	Returns 10 because there are no rows above this row so 2 of the 3 values are missing (=NULL).
abc	0	
8	8	
def	0	
xyz	0	
9	9	

See also:

- [Only - chart function \(page 156\)](#)

RangeSkew

RangeSkew() returns the value corresponding to the skewness of a range of numbers.

Syntax:

```
RangeSkew (first_expr{, Expression})
```

Return data type: numeric

Arguments:

The argument expressions of this function may contain inter-record functions with a third optional parameter, which in themselves return a range of values.

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

If no numeric value is found, NULL is returned.

Examples and results:

Examples	Results
rangeskew (1,2,4)	Returns 0.93521952958283
rangeskew (above (count(x),0,3))	Returns a sliding skewness of the result of the inner count(x) expression calculated on the current row and two rows above the current row.

Data used in example:

CustID	RangeSkew(SalesValue, Above(SalesValue, Above(Above(SalesValue))))
1-20	-, -, 0.5676, 0.8455, 1.0127, -0.8741, 1.7243, -1.7186, 1.5518, 1.4332, 0, 1.1066, 1.3458, 1.5636, 1.5439, 0.6952, -0.3766

```
SalesTable:  
LOAD recno() as CustID, * inline [  
SalesValue
```

101
163
126
139
167
86
83
22
32
70
108
124
176
113
95
32
42
92
61
21
] ;

See also:

▫ [Skew - chart function \(page 219\)](#)

RangeStdev

RangeStdev() finds the standard deviation of a range of numbers.

Syntax:

```
RangeStdev (first_expr{, Expression})
```

Return data type: numeric

Arguments:

The argument expressions of this function may contain inter-record functions with a third optional parameter, which in themselves return a range of values.

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

If no numeric value is found, NULL is returned.

Examples and results:

5 Functions in scripts and chart expressions

Examples	Results
RangeStdev (1,2,4)	Returns 1.5275252316519
RangeStdev (null())	Returns NULL
RangeStdev (above (count(x),0,3))	Returns a sliding standard deviation of the result of the inner count(x) expression calculated on the current row and two rows above the current row.

Data used in example:

CustID	RangeStdev(SalesValue, Above(SalesValue, Above(Above(SalesValue)))
1-20	-,43.841, 34.192, 18.771, 20.953, 41.138, 47.655, 36.116, 32.716, 25.325, 38,000, 27.737, 35.553, 33.650, 42.532, 33.858, 32.146, 25.239, 35.595

```
SalesTable:  
LOAD recno() as CustID, * inline [  
SalesValue  
101  
163  
126  
139  
167  
86  
83  
22  
32  
70  
108  
124  
176  
113  
95  
32  
42  
92  
61  
21  
] ;
```

See also:

▢ [Stdev - chart function \(page 221\)](#)

RangeSum

RangeSum() returns the sum of a range of values. All non-numeric values are treated as 0, unlike the **+** operator.

5 Functions in scripts and chart expressions

Syntax:

```
RangeSum (first_expr {,Expression})
```

Return data type: numeric

Arguments:

The argument expressions of this function may contain inter-record functions with a third optional parameter, which in themselves return a range of values.

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

The **RangeSum** function treats all non-numeric values as 0, unlike the **+** operator.

Examples and results:

Examples	Results
RangeSum (1,2,4)	Returns 7
RangeSum (5,'abc')	Returns 5
RangeSum (null())	Returns 0

Example with expression:

```
RangeSum (Above(Sum(MyField),0,3))
```

Returns the sum of the three results of the **Sum(MyField)** function evaluated over the current row and two rows above the current row.

Data used in examples:



*Disable sorting of **MyField** to ensure that example works as expected.*

MyField	Sum(MyField)	RangeSum(Above(Sum(MyField),0,3))
10	10	10
abc	0	10

MyField	Sum(MyField)	RangeSum(Above(Sum(MyField),0,3))
8	8	18
def	0	8
xyz	0	8
9	9	9

See also:

- ▢ [Sum - chart function \(page 158\)](#)

RangeTextCount

RangeTextCount() returns the number of text values in an expression or field.

Syntax:

```
RangeTextCount (first_expr {, Expression})
```

Return data type: integer

Arguments:

The argument expressions of this function may contain inter-record functions with a third optional parameter, which in themselves return a range of values.

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Examples and results:

Examples	Results
RangeTextCount (1,2,4)	Returns 0
RangeTextCount (5, 'abc')	Returns 1
RangeTextCount (null())	Returns 0

Example with expression:

```
RangeTextCount (Above(MaxString(MyField),0,3))
```

5 Functions in scripts and chart expressions

Returns the number of text values within the three results of the **MaxString(MyField)** function evaluated over the current row and two rows above the current row.

Data used in examples:



*Disable sorting of **MyField** to ensure that example works as expected.*

MyField	MaxString(MyField)	RangeTextCount(Above(Sum(MyField),0,3))
10	10	0
abc	abc	1
8	8	1
def	def	2
xyz	xyz	2
9	9	2

See also:

- [TextCount - chart function \(page 173\)](#)

RangeXIRR

This script function returns the internal rate of return for a schedule of cash flows that is not necessarily periodic. To calculate the internal rate of return for a series of periodic cash flows, use the **RangeIRR** function.

Syntax:

```
RangeXIRR(value, date { ,value, date} )
```

Arguments:

Argument	Description
value	A cash flow or a series of cash flows that corresponds to a schedule of payments in dates. Each value may be a single value or a range of values as returned by an inter-record function with a third optional parameter. Text values, NULL values and missing values are disregarded. All payments are discounted based on a 365-day year. The series of values must contain at least one positive and one negative value.
date	A payment date or a schedule of payment dates that corresponds to the cash flow payments.

Examples:

`RangeXIRR(-2500, '2008-01-01', 2750, '2008-09-01')` returns 0,1532

```
RangeXIRR (above(sum(value), 0, 10), above(date, 0, 10))
RangeXIRR(above(total value,0,rowno(total)),
above(total date,0,rowno(total)))
```

See also:

▫ [RangeIRR \(page 483\)](#)

RangeXNPV

This script function returns the net present value for a schedule of cash flows that is not necessarily periodic. The result has a default number format of money. To calculate the net present value for a series of periodic cash flows, use the **RangeNPV** function.

Syntax:

```
RangeXNPV(rate, value, date { ,value, date} )
```

Arguments:

Argument	Description
rate	The interest rate per period.
value	A cash flow or a series of cash flows that corresponds to a schedule of payments in dates. Each value may be a single value or a range of values as returned by an inter-record function with a third optional parameter. Text values, NULL values and missing values are disregarded. All payments are discounted based on a 365-day year. The series of values must contain at least one positive and one negative value.
date	A payment date or a schedule of payment dates that corresponds to the cash flow payments.

Examples:

`RangeXNPV(0.1, -2500, '2008-01-01', 2750, '2008-09-01')` returns 80,25

```
RangeXNPV(0.1, above(sum(value), 0, 10), above(date, 0, 10))
RangeXNPV(0.1, above(total value,0,rowno(total)),
```

5.20 Ranking functions in charts

These functions can only be used in chart expressions.



Suppression of zero values is automatically disabled when these functions are used. NULL values are disregarded.

Rank

Rank() evaluates the rows of the chart in the expression, and for each row, displays the relative position of the value of the dimension evaluated in the expression. When evaluating the expression, the function compares the result with the result of the other rows containing the current column segment and returns the ranking of the current row within the segment.

```
Rank - chart function([TOTAL [<fld {, fld}>]] expr[, mode[, fmt]])
```

VRank

VRank() performs the same function as **Rank** function. You may use either.

```
VRank - chart function([TOTAL [<fld {, fld}>]] expr[, mode[, fmt]])
```

Rank - chart function

Rank() evaluates the rows of the chart in the expression, and for each row, displays the relative position of the value of the dimension evaluated in the expression. When evaluating the expression, the function compares the result with the result of the other rows containing the current column segment and returns the ranking of the current row within the segment.

For charts other than tables, the current column segment is defined as it appears in the chart's straight table equivalent.

Syntax:

```
Rank ([TOTAL [<fld {, fld}>]] expr[, mode[, fmt]])
```

Return data type: dual

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
mode	Specifies the number representation of the function result.
fmt	Specifies the text representation of the function result.
TOTAL	If the chart is one-dimensional, or if the expression is preceded by the TOTAL qualifier, the function is evaluated across the entire column. If the table or table equivalent has multiple vertical dimensions, the current column segment will include only rows with the same values as the current row in all dimension columns except for the column showing the last dimension in the inter-field sort order. The TOTAL qualifier may be followed by a list of one or more field names within angle brackets <fld>. These field names should be a subset of the chart dimension variables.

5 Functions in scripts and chart expressions

The ranking is returned as a dual value, which in the case when each row has a unique ranking, is an integer between 1 and the number of rows in the current column segment.

In the case where several rows share the same ranking, the text and number representation can be controlled with the **mode** and **fmt** parameters.

mode

The second argument, **mode**, can take the following values:

Value	Description
0 (default)	If all ranks within the sharing group fall on the low side of the middle value of the entire ranking, all rows get the lowest rank within the sharing group. If all ranks within the sharing group fall on the high side of the middle value of the entire ranking, all rows get the highest rank within the sharing group. If ranks within the sharing group span over the middle value of the entire ranking, all rows get the value corresponding to the average of the top and bottom ranking in the entire column segment.
1	Lowest rank on all rows.
2	Average rank on all rows.
3	Highest rank on all rows.
4	Lowest rank on first row, then incremented by one for each row.

fmt

The third argument, **fmt**, can take the following values:

Value	Description
0 (default)	Low value - high value on all rows (for example 3 - 4).
1	Low value on all rows.
2	Low value on first row, blank on the following rows.

The order of rows for **mode** 4 and **fmt** 2 is determined by the sort order of the chart dimensions.

Examples and results:

Create two visualizations from the dimensions Product and Sales and another from Product and UnitSales. Add measures as shown in the following table.

Examples	Results
Create a table with	The result depends on the sort order of the dimensions. If the tables is sorted on Customer, the table lists all the values of Sales for Astrida, then Betacab, and so on. The results for

5 Functions in scripts and chart expressions

Examples	Results
and sales and the measure Rank (Sales))	<p>Rank(Sales) will show 10 for the Sales value 12, 9 for the Sales value 13, and so on, with the rank value of 1 returned for the Sales value 78. Note that even though there are 12 Sales values, only 11 rows are shown because two values of Sales are the same (78). The next column segment begins with Betacab, for which the first value of Sales in the segment is 12. The rank value of Rank(Sales) for this is given as 11.</p> <p>If the table is sorted on Sales, the column segments consist of the values of Sales and the corresponding Customer. Because there are two Sales values of 12 (for Astrida and Betacab), the value of Rank(Sales) for that column segment is 1-2, for each value of Customer. This is because there are two values of Customer for the Sales value 12. If there had been 4 values, the result would be 1-4, for all rows. This shows what the result looks like for the default value (0) of the argument fmt.</p>
Replace the dimension Customer with Product and add the measure Rank (Sales,1,2)	<p>This returns 1 on the first row on each column segment and leaves all other rows blank, because arguments mode and fmt are set to 1 and 2 respectively.</p>

Data used in examples:

ProductData:

```
Load * inline [
Customer|Product|UnitSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD|0|25
Canutility|AA|8|15
Canutility|CC|0|19
] (delimiter is '|');
```

Sales2013:

```
crosstable (Month, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

See also:

- ▢ *VRank - chart function (page 508)*
- ▢ *Sum - chart function (page 158)*

VRank - chart function

VRank() performs the same function as **Rank** function. You may use either.

Syntax:

```
VRank ([TOTAL [<fld {,fld}>]] expr[, mode[, fmt]])
```

Return data type: dual

See also:

- ▢ *Rank - chart function (page 505)*

5.21 Statistical distribution functions

The statistical distribution functions described below are all implemented in Qlik Sense using the Cephes library. For references and details on algorithms used, accuracy etc, see <http://www.netlib.org/cephes/>. The Cephes function library is used by permission.

All functions can be used in both the data load script and in chart expressions.

Statistical distribution functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

CHIIDIST

This function returns the one-tailed probability of the chi2 distribution. The chi2distribution is associated with a chi2 test.

```
CHIIDIST (value, degrees_freedom)
```

CHIINV

This function returns the inverse of the one-tailed probability of the chi2 distribution.

```
CHIINV (prob, degrees_freedom)
```

NORMDIST

This function returns the cumulative normal distribution for the specified mean and standard deviation. If

5 Functions in scripts and chart expressions

mean = 0 and standard_dev = 1, the function returns the standard normal distribution.

```
NORMDIST (value, mean, standard_dev)
```

NORMINV

This function returns the inverse of the normal cumulative distribution for the specified mean and standard deviation.

```
NORMINV (prob, mean, standard_dev)
```

TDIST

This function returns the probability for the student t-distribution where a numeric value is a calculated value of t for which the probability is to be computed.

```
TDIST (value, degrees_freedom, tails)
```

TINV

This function returns the t-value of the student's t-distribution as a function of the probability and the degrees of freedom.

```
TINV (prob, degrees_freedom)
```

FDIST

This function returns the F-distribution.

```
FDIST (value, degrees_freedom1, degrees_freedom2)
```

FINV

This function returns the inverse of the F-distribution.

```
FINV (prob, degrees_freedom1, degrees_freedom2)
```

CHIDIST

This function returns the one-tailed probability of the chi2 distribution. The chi2distribution is associated with a chi2 test.

Syntax:

```
CHIDIST (value, degrees_freedom)
```

This function is related to the **CHIINV** function in the following way:

If $prob = CHIDIST(value, df)$, then $CHIINV(prob, df) = value$.

Arguments:

5 Functions in scripts and chart expressions

Argument	Description
value	The value at which you want to evaluate the distribution. The value must not be negative.
degrees_freedom	A positive integer stating the number of degrees of freedom. Both arguments must be numeric, else NULL will be returned.

Examples and results:

Example	Result
CHIDIST(8, 15)	Returns 0.9237827

CHIINV

This function returns the inverse of the one-tailed probability of the chi2 distribution.

Syntax:

```
CHIINV (prob, degrees_freedom)
```

This function is related to the **CHIDIST** function in the following way:

If $prob = chidist(value, df)$, then $chiinv(prob, df) = value$.

Arguments:

Argument	Description
prob	A probability associated with the chi2 distribution. It must be a number between 0 and 1
degrees_freedom	An integer stating the number of degrees of freedom.

Limitations:

All arguments must be numeric, else NULL will be returned.

Examples and results:

Example	Result
CHIINV(0.9237827, 15)	Returns 8.0000001

FDIST

This function returns the F-distribution.

Syntax:

5 Functions in scripts and chart expressions

```
FDIST(value, degrees_freedom1, degrees_freedom2)
```

This function is related to the **FINV** function in the following way:

If $\text{prob} = \text{FDIST}(\text{value}, \text{df1}, \text{df2})$, then $\text{FINV}(\text{prob}, \text{df1}, \text{df2}) = \text{value}$.

Arguments:

Argument	Description
value	The value at which you want to evaluate the distribution. Value must not be negative.
degrees_freedom1	A positive integer stating the number of numerator degrees of freedom.
degrees_freedom2	A positive integer stating the number of denominator degrees of freedom.

Limitations:

All arguments must be numeric, else NULL will be returned.

Examples and results:

Example	Result
FDIST(15, 8, 6)	Returns 0.0019369

FINV

This function returns the inverse of the F-distribution.

Syntax:

```
FINV (prob, degrees_freedom1, degrees_freedom2)
```

This function is related to the **FDIST** function in the following way:

If $\text{prob} = \text{fdist}(\text{value}, \text{df1}, \text{df2})$, then $\text{finv}(\text{prob}, \text{df1}, \text{df2}) = \text{value}$.

Arguments:

Argument	Description
prob	A probability associated with the F-distribution and must be a number between 0 and 1.
degrees_freedom	An integer stating the number of degrees of freedom.

Limitations:

All arguments must be numeric, else NULL will be returned.

Examples and results:

Example	Result
FINV(0.0019369, 8, 5)	Returns 15.0000197

NORMDIST

This function returns the cumulative normal distribution for the specified mean and standard deviation. If mean = 0 and standard_dev = 1, the function returns the standard normal distribution.

Syntax:

```
NORMDIST (value, mean, standard_dev)
```

This function is related to the **NORMINV** function in the following way:

If $\text{prob} = \text{normdist}(\text{value}, m, \text{sd})$, then $\text{norminv}(\text{prob}, m, \text{sd}) = \text{value}$.

Arguments:

Argument	Description
value	The value at which you want to evaluate the distribution.
mean	A value stating the arithmetic mean for the distribution.
standard_dev	A positive value stating the standard deviation of the distribution.

Limitations:

All arguments must be numeric, else NULL will be returned.

Examples and results:

Example	Result
NORMDIST(0.5, 0, 1)	Returns 0.691462

NORMINV

This function returns the inverse of the normal cumulative distribution for the specified mean and standard deviation.

Syntax:

```
NORMINV (prob, mean, standard_dev)
```


5 Functions in scripts and chart expressions

This function is related to the **NORMDIST** function in the following way:

If $\text{prob} = \text{NORMDIST}(\text{value}, m, \text{sd})$, then $\text{NORMINV}(\text{prob}, m, \text{sd}) = \text{value}$.

Arguments:

Argument	Description
prob	A probability associated with the normal distribution. It must be a number between 0 and 1.
mean	A value stating the arithmetic mean for the distribution.
standard_dev	A positive value stating the standard deviation of the distribution.

Limitations:

All arguments must be numeric, else NULL will be returned.

Examples and results:

Example	Result
<code>NORMINV(0.6914625, 0, 1)</code>	Returns 0.4999717

TDIST

This function returns the probability for the student t-distribution where a numeric value is a calculated value of t for which the probability is to be computed.

Syntax:

```
TDIST (value, degrees_freedom, tails)
```

This function is related to the **TINV** function in the following way:

If $\text{prob} = \text{tdist}(\text{value}, \text{df}, 2)$, then $\text{tinv}(\text{prob}, \text{df}) = \text{value}$.

Arguments:

Argument	Description
value	The value at which you want to evaluate the distribution and must not be negative.
degrees_freedom	A positive integer stating the number of degrees of freedom.
tails	Must be either 1 (one-tailed distribution) or 2 (two-tailed distribution).

Limitations:

5 Functions in scripts and chart expressions

All arguments must be numeric, else NULL will be returned.

Examples and results:

Example	Result
chdist(1, 30, 2)	Returns 0.3253086

TINV

This function returns the t-value of the student's t-distribution as a function of the probability and the degrees of freedom.

Syntax:

```
TINV (prob, degrees_freedom)
```

This function is related to the **TDIST** function in the following way:

If $prob = tdist(value, df, 2)$, then $tinv(prob, df) = value$.

Arguments:

Argument	Description
prob	A two-tailed probability associated with the t-distribution. It must be a number between 0 and 1.
degrees_freedom	An integer stating the number of degrees of freedom.

Limitations:

All arguments must be numeric, else NULL will be returned.

Examples and results:

Example	Result
TINV(0.3253086, 30)	Returns 1

5.22 String functions

This section describes functions for handling and manipulating strings. In the functions below, the parameters are expressions where **s** should be interpreted as a string.

All functions can be used in both the data load script and in chart expressions, except for **Evaluate** which can only be used in the data load script.

String functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

ApplyCodepage

Applies a different codepage to the field or text stated in the expression. The codepage must be in number format.

```
ApplyCodepage(text, codepage)
```

Capitalize

This function returns the string **s** with all words capitalized.

```
Capitalize ( s )
```

Chr

This function returns the ASCII character corresponding to number **n**. The result is a string.

```
Chr ( n )
```

Evaluate

This script function returns the evaluated result of the expression if the text string **s** can be evaluated as a valid Qlik Sense expression. If **s** is not a valid expression, NULL is returned.

```
Evaluate ( s )
```

FindOneOf

This function returns the position of the **n**:the occurrence in the string **text** of any of the characters found in the string **characterset**. If **n** is omitted, the position of the first occurrence is returned. If there no matching string is found, **0** is returned.

```
FindOneOf (text , characterset [ , n])
```

Hash128

This function returns a 128-bit hash of the combined input expression values. The result is a string.

```
Hash128 (expression {, expression})
```

Hash160

This function returns a 160-bit hash of the combined input expression values. The result is a string.

```
Hash160 (expression {, expression})
```

Hash256

This function returns a 256-bit hash of the combined input expression values. The result is a string.

```
Hash256 (expression {, expression} )
```

Index

5 Functions in scripts and chart expressions

This function returns the starting position of the **n**:th occurrence of substring **s2** in string **s1**. If **n** is omitted, the first occurrence is assumed. If **n** is negative, the search is made starting from the end of string **s1**. The result is an integer. The positions in the string are numbered from **1** and up.

```
Index ( s1 , s2[ , n] )
```

KeepChar

This function returns the string **s1** less all characters not contained in string **s2**.

```
KeepChar (s1 , s2)
```

Left

This function returns a string consisting of the first **n** characters of **s**.

```
Left ( s , n )
```

Len

This function returns the length of string **s**. The result is an integer.

```
Len ( s )
```

Lower

This function forces lower case for all data in the expression.

```
Lower ( textexpression )
```

LTrim

This function returns the string **s** trimmed of any leading spaces.

```
LTrim (s)
```

Mid

This function returns the string starting at character **n1** with the length of **n2** characters. If **n2** is omitted, the function returns the rightmost part of the string starting at character **n1**. The positions in the string are numbered from **1** and up.

```
Mid (s, n1[, n2 ])
```

Ord

This function returns the ASCII number of first character of string **s**. The result is an integer.

```
Ord ( s )
```

PurgeChar

This function returns the string **s1** less all characters contained in string **s2**.

```
PurgeChar (s1, s2)
```

Repeat

This function forms a string consisting of the string **s** repeated **n** times.

5 Functions in scripts and chart expressions

```
Repeat ( s, n )
```

Replace

This function returns a string after replacing all occurrences of a given substring within the string **s** with another substring. The function is non-recursive and works from left to right.

```
Replace (s, fromstring ,tostring)
```

Right

This function returns a string consisting of the last **n** characters of **s**.

```
Right ( s , n )
```

RTrim

This function returns the string **s** trimmed of any trailing spaces.

```
RTrim ( s )
```

SubField

In its three-parameter version, this function returns a given substring from a larger string **s** with delimiter '**delimiter**'. **Index** is an optional integer denoting which of the substrings should be returned. If **index** is omitted when **subfield** is used in a field expression in a **LOAD** statement, the **subfield** function will cause the **LOAD** statement to automatically generate one full record of input data for each sub-string that can be found in **s**.

In its two-parameter version, the **subfield** function generates one record for each substring that can be taken from a larger string **s** with the delimiter '**delimiter**'. If several **subfield** functions are used in the same **LOAD** statement, the Cartesian product of all combinations will be generated.

```
SubField ( s, 'delimiter' [ , index ] )
```

SubStringCount

This function returns the number of times the string substring appears within the string text. The result is an integer. If there is no match, 0 is returned.

```
SubStringCount ( text , substring)
```

TextBetween

This function returns the text between the **n**:th occurrence of **beforetext** and the immediately following occurrence of **aftertext** within the string **s**.

```
TextBetween (s , beforetext , aftertext [, n ])
```

Trim

This function returns the string **s** trimmed of any leading and trailing spaces.

```
Trim ( s )
```

Upper

5 Functions in scripts and chart expressions

This function forces upper case for all data in the expression.

```
Upper ( textexpression )
```

Capitalize

This function returns the string **s** with all words capitalized.

Syntax:

```
Capitalize( s )
```

Examples and results:

Example	Result
capitalize ('my little pony')	Returns 'My Little Pony'
capitalize ('AA bb cc Dd')	Returns 'Aa Bb Cc Dc'

Chr

This function returns the ASCII character corresponding to number **n**. The result is a string.

Syntax:

```
Chr( n )
```

Examples and results:

Example	Result
Chr(65)	Returns the string 'A'

Evaluate

This script function returns the evaluated result of the expression if the text string **s** can be evaluated as a valid Qlik Sense expression. If **s** is not a valid expression, NULL is returned.

Syntax:

```
Evaluate( s )
```



This string function can not be used in chart expressions.

Examples and results:

Example	Result
evaluate (5 * 8)	Returns '40'

FindOneOf

This function returns the position of the **n**:the occurrence in the string **text** of any of the characters found in the string **characterset**. If **n** is omitted, the position of the first occurrence is returned. If there no matching string is found, **0** is returned.

Syntax:

```
FindOneOf(text , characterset [ , n])
```

Argument	Description
text	The original string.
characterset	A set of characters to evaluate.
n	The number of occurrences of the character to evaluate.

Examples and results:

Example	Result
FindOneOf('my example text string', 'et%s')	Returns '4'
FindOneOf('my example text string', 'et%s', 3)	Returns '12'
FindOneOf('my example text string', 'x%&')	Returns '0'

Hash128

This function returns a 128-bit hash of the combined input expression values. The result is a string.

Syntax:

```
Hash128(expression {, expression})
```

Example:

```
Hash128 ( 'abc', 'xyz', '123' )  
Hash128 ( Region, Year, Month )
```

Hash160

This function returns a 160-bit hash of the combined input expression values. The result is a string.

Syntax:

```
Hash160(expression {, expression})
```

Example:

Hash160 (Region, Year, Month)

Hash256

This function returns a 256-bit hash of the combined input expression values. The result is a string.

Syntax:

```
Hash256(expression {, expression} )
```

Example:

Hash256 (Region, Year, Month)

Index

This function returns the starting position of the **n**:th occurrence of substring **s2** in string **s1**. If **n** is omitted, the first occurrence is assumed. If **n** is negative, the search is made starting from the end of string **s1**. The result is an integer. The positions in the string are numbered from **1** and up.

Syntax:

```
Index( s1 , s2 [ , n ] )
```

Examples and results:

For a more complicated example, see the **index** function below.

Example	Result
Index('abcdefg', 'cd')	Returns 3
Index('abcdefg', 'b', 2)	Returns 6
Index('abcdefg', 'b', -2)	Returns 2
Left(Date, Index(Date, '-') -1) where Date = 1997-07-14	Returns 1997
Mid(Date, Index(Date, '-' , 2) -2, 2) where Date = 1997-07-14	Returns 07

KeepChar

This function returns the string **s1** less all characters not contained in string **s2**.

Syntax:

```
KeepChar( s1 , s2)
```

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
KeepChar ('a1b2c3', '123')	Returns '123'

Left

This function returns a string consisting of the first **n** characters of **s**.

Syntax:

```
Left( s , n )
```

Examples and results:

For a more complicated example, see the **index** function.

Example	Result
Left('abcdef', 3)	Returns 'abc'
Left(Date, 4) where Date = 1997-07-14	Returns '1997'

See also:

▢ [Index \(page 520\)](#)

Len

This function returns the length of string **s**. The result is an integer.

Syntax:

```
Len( s )
```

Examples and results:

Example	Result
Len(Name) where Name = 'Peter'	Returns '5'

Lower

This function forces lower case for all data in the expression.

Syntax:

```
Lower( textexpression )
```

Examples and results:

5 Functions in scripts and chart expressions

Example	Result
Lower('abcD')	Returns 'abcd'

LTrim

This function returns the string **s** trimmed of any leading spaces.

Syntax:

```
LTrim( s )
```

Examples and results:

Example	Result
LTrim(' abc')	Returns 'abc'
LTrim('abc ')	Returns 'abc'

Mid

This function returns the string starting at character **n1** with the length of **n2** characters. If **n2** is omitted, the function returns the rightmost part of the string starting at character **n1**. The positions in the string are numbered from **1** and up.

Syntax:

```
Mid(s, n1[, n2 ])
```

Examples and results:

For a more complicated example, see the **index** function.

Example	Result
Mid('abcdef',3)	Returns 'cdef'
Mid('abcdef',3, 2)	Returns 'cd'
Mid(Date,3) where Date = 970714	Returns '0714'
Mid(Date,3,2) where Date = 970714	Returns '07'

See also:

▢ [Index \(page 520\)](#)

Ord

This function returns the ASCII number of first character of string **s**. The result is an integer.

Syntax:

```
Ord( s )
```

Examples and results:

Example	Result
ord('A')	Returns the number '65'

PurgeChar

This function returns the string **s1** less all characters contained in string **s2**.

Syntax:

```
PurgeChar(s1, s2)
```

Examples and results:

Example	Result
PurgeChar ('a1b2c3', '123')	Returns 'abc'

Repeat

This function forms a string consisting of the string **s** repeated **n** times.

Syntax:

```
Repeat( s, n )
```

Examples and results:

Example	Result
Repeat(' * ', rating) when rating = 4	Returns '****'

Replace

This function returns a string after replacing all occurrences of a given substring within the string **s** with another substring. The function is non-recursive and works from left to right.

Syntax:

5 Functions in scripts and chart expressions

```
Replace(s, fromstring ,tostring)
```

Arguments:

Argument	Description
s	The original string.
fromstring	A string which may occur one or more times within string.
tostring	The string which will replace all occurrences of fromstring within the string.

Examples and results:

Example	Result
Replace('abcde', 'cc', 'xyz')	Returns 'abxyzde'

Right

This function returns a string consisting of the last **n** characters of **s**.

Syntax:

```
Right( s , n )
```

Examples and results:

Example	Result
Right('abcdef', 3)	Returns 'def'
Right(Date,2) where Date = 1997-07-14	Returns '14'

RTrim

This function returns the string **s** trimmed of any trailing spaces.

Syntax:

```
RTrim( s )
```

Examples and results:

Example	Result
RTrim(' abc')	Returns 'abc'
RTrim('abc ')	Returns 'abc'

SubField

In its three-parameter version, this function returns a given substring from a larger string **s** with delimiter '**delimiter**'. **Index** is an optional integer denoting which of the substrings should be returned. If **index** is omitted when **subfield** is used in a field expression in a **LOAD** statement, the **subfield** function will cause the **LOAD** statement to automatically generate one full record of input data for each sub-string that can be found in **s**.

In its two-parameter version, the **subfield** function generates one record for each substring that can be taken from a larger string **s** with the delimiter '**delimiter**'. If several **subfield** functions are used in the same **LOAD** statement, the Cartesian product of all combinations will be generated.

Syntax:

```
SubField ( s, 'delimiter' [ , index ] )
```

Examples and results:

Example	Result
SubField(S, ';' ,2)	Returns 'cde' if S is 'abc;cde;efg'
SubField(S, ';' ,1)	Returns NULL if S is an empty string
SubField(S, ';' ,1)	Returns an empty string if S is ''

SubStringCount

This function returns the number of times the string substring appears within the string text. The result is an integer. If there is no match, 0 is returned.

Syntax:

```
SubStringCount( text , substring)
```

Examples and results:

Example	Result
SubStringCount ('abcdefgcdxyz', 'cd')	Returns '2'

TextBetween

This function returns the text between the **n**:th occurrence of **beforetext** and the immediately following occurrence of **aftertext** within the string **s**.

Syntax:

```
TextBetween(s , beforetext , aftertext [ , n ])
```

Examples and results:

Example	Result
<code>TextBetween('<abc>', '<', '>')</code>	Returns 'abc'
<code>TextBetween('<abc><de>', '<', '>', 2)</code>	Returns 'de'

Trim

This function returns the string **s** trimmed of any leading and trailing spaces.

Syntax:

```
Trim( s )
```

Examples and results:

Example	Result
<code>Trim(' abc ')</code>	Returns 'abc'
<code>Trim('abc ')</code>	Returns 'abc'
<code>Trim(' abc ')</code>	Returns 'abc'

Upper

This function forces upper case for all data in the expression.

Syntax:

```
Upper( textexpression )
```

Examples and results:

Example	Result
<code>Upper(' abcd')</code>	Returns 'ABCD'

5.23 System functions

System functions provide functions for accessing system, device and Qlik Sense app properties.

System functions overview

Some of the functions are described further after the overview. For those functions, you can click the function name in the syntax to immediately access the details for that specific function.

Author()

5 Functions in scripts and chart expressions

This function returns a string containing the author property of the current app. It can be used in both the data load script and in a chart expression.



Author property can not be set in the current version of Qlik Sense. If you migrate a QlikView document, the author property will be retained.

ClientPlatform()

This function returns the user agent string of the client browser. It can be used in both the data load script and in a chart expression..

Example:

```
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.114 Safari/537.36
```

ComputerName

This function returns a string containing the name of the computer as returned by the operating system. It can be used in both the data load script and in a chart expression..

```
ComputerName ( )
```

DocumentName

This function returns a string containing the name of the current Qlik Sense app, without path but with extension. It can be used in both the data load script and in a chart expression.

```
DocumentName ( )
```

DocumentPath

This function returns a string containing the full path to the current Qlik Sense app. It can be used in both the data load script and in a chart expression.

```
DocumentPath ( )
```



This function is not supported in standard mode.

DocumentTitle

This function returns a string containing the title of the current Qlik Sense app. It can be used in both the data load script and in a chart expression.

```
DocumentTitle ( )
```

OSUser

This function returns a string containing the name of the current user as returned by the operating system. It can be used in both the data load script and in a chart expression..

```
OSUser ( )
```

QlikViewVersion

5 Functions in scripts and chart expressions

This function returns the full Qlik Sense version and build number (e.g. 7.52.3797.0409.3) as a string.

```
QlikViewVersion ( )
```

ReloadTime

This function returns a timestamp for when the last data load finished. It can be used in both the data load script and in a chart expression..

```
ReloadTime ( )
```

GetExtendedProperty

This function returns the value of a named extended property in the sheet object with the given object ID. If **objectid** is not given, the sheet object containing the expression will be used. An extended property is defined for the extension object in its definition file.

```
GetExtendedProperty - chart function (name[, objectid])
```

GetObjectField

This function returns the name of the dimension. **Index** is an optional integer denoting which of the used dimensions that should be returned.

```
GetObjectField - chart function ([index])
```

GetRegistryString

This function returns the value of a key in the Windows registry. It can be used in both the data load script and in a chart expression..

```
GetRegistryString (path, key)
```

GetCollationLocale

This script function returns the culture name of the collation locale that is used. If the variable CollationLocale has not been set, the actual user machine locale is returned..

```
GetCollationLocale ( )
```

See also:

▢ [GetFolderPath \(page 413\)](#)

GetExtendedProperty - chart function

This function returns the value of a named extended property in the sheet object with the given object ID. If **objectid** is not given, the sheet object containing the expression will be used. An extended property is defined for the extension object in its definition file.

Syntax:

```
GetExtendedProperty (name[, objectid])
```

Example:


```
GetExtendedProperty ('Greeting')
```

GetObjectField - chart function

This function returns the name of the dimension. **Index** is an optional integer denoting which of the used dimensions that should be returned.

Syntax:

```
GetObjectField ([index])
```

Example:

```
getObjectField(2)
```

QlikViewVersion

This function returns the full Qlik Sense version and build number (e.g. 7.52.3797.0409.3) as a string.

Syntax:

```
QlikViewVersion()
```

5.24 Table functions

The table functions return information about the data table which is currently being read. If no table name is specified and the function is used within a **LOAD** statement, the current table is assumed.

All functions can be used in the data load script, while only **NoOfRows** can be used in a chart expression.

Table functions overview

Some of the functions are described further after the overview. For those functions, you can click the function name in the syntax to immediately access the details for that specific function.

FieldName

This script function returns the name of the field with the specified number within a previously loaded table. If the function is used within a **LOAD** statement, it must not reference the table currently being loaded.

```
FieldName (nr , 'TableName')
```

FieldNumber

This script function returns the number of a specified field within a previously loaded table. If the function is used within a **LOAD** statement, it must not reference the table currently being loaded.

```
FieldNumber ('field ' , 'TableName')
```

NoOfFields

This script function returns the number of fields in a previously loaded table. If the function is used within a

5 Functions in scripts and chart expressions

LOAD statement, it must not reference the table currently being loaded.

```
NoOfFields ([ 'TableName ' ])
```

NoOfRows

This function returns the number of rows (records) in a previously loaded table. If the function is used within a **LOAD** statement, it must not reference the table currently being loaded.

```
NoOfRows ([ 'TableName ' ])
```

NoOfTables

This script function returns the number of tables previously loaded.

```
NoOfTables ()
```

TableName

This script function returns the name of the table with the specified number.

```
TableName ([ 'TableNumber ' ])
```

TableNumber

This script function returns the number of the specified table.

```
TableNumber ([ 'TableName ' ])
```

FieldName

This script function returns the name of the field with the specified number within a previously loaded table. If the function is used within a **LOAD** statement, it must not reference the table currently being loaded.

Syntax:

```
FieldName(nr , 'TableName')
```

Example:

```
LET a = FieldName(4,'tab1');  
T1:  
LOAD a, b, c, d from abc.csv  
T2:  
LOAD FieldName (2, 'T1') Autogenerate 1;
```

FieldNumber

This script function returns the number of a specified field within a previously loaded table. If the function is used within a **LOAD** statement, it must not reference the table currently being loaded.

Syntax:

```
FieldNumber('field ' , 'TableName')
```

Example:

5 Functions in scripts and chart expressions

```
LET a = FieldNumber('Customer','tab1');
T1:
LOAD a, b, c, d from abc.csv
T2:
LOAD FieldNumber ('b', 'T1') Autogenerate 1;
```

NoOfFields

This script function returns the number of fields in a previously loaded table. If the function is used within a **LOAD** statement, it must not reference the table currently being loaded.

Syntax:

```
NoOfFields([ 'TableName ' ])
```

Example:

```
LET a = NoOfFields('tab1');
LOAD *, NoOfFields( ) from abc.csv;
```

NoOfRows

This function returns the number of rows (records) in a previously loaded table. If the function is used within a **LOAD** statement, it must not reference the table currently being loaded.

Syntax:

```
NoOfRows([ 'TableName ' ])
```

Example:

```
LET a = NoOfRows('tab1');
LOAD * from abc.csv where NoOfRows( )<30;
```

5.25 Trigonometric and hyperbolic functions

This section describes functions for performing trigonometric and hyperbolic operations. In the functions below, the parameters are expressions where **x** should be interpreted as a real valued number.

All angles are measured in radians.

All functions can be used in both the data load script and in chart expressions.

cos

Cosine of **x**. The result is a number between -1 and 1.

```
cos ( x )
```

acos

Arcus cosine of **x**. The function is only defined if $-1 \leq x \leq 1$. The result is a number between 0 and π .

5 Functions in scripts and chart expressions

```
acos( x )
```

sin

Sine of **x**. The result is a number between -1 and 1.

```
sin( x )
```

asin

Arcus sine of **x**. The function is only defined if $-1 \leq x \leq 1$. The result is a number between $-\pi/2$ and $\pi/2$.

```
asin( x )
```

tan

Tangent of **x**. The result is a number.

```
tan( x )
```

atan

Arcus tangent of **x**. The result is a number between $-\pi/2$ and $\pi/2$.

```
atan( x )
```

atan2

Two-dimensional generalization of the arcus tangent function. Returns the angle between the origin and the point represented by the coordinates **x** and **y**. The result is a number between $-\pi$ and $+\pi$.

```
atan2( y, x )
```

cosh

Cosine hyperbolicus of **x**. The result is a positive number.

```
cosh( x )
```

sinh

Sine hyperbolicus of **x**. The result is a number.

```
sinh( x )
```

tanh

Tangent hyperbolicus of **x**. The result is a number.

```
tanh( x )
```

5 File system access restriction

For security reasons, Qlik Sense in standard mode does not support absolute or relative paths in the data load script or functions and variables that expose the file system.

However, since absolute and relative paths were supported in QlikView, it is possible to disable standard mode and use legacy mode in order to reuse QlikView load scripts.



Disabling standard mode can create a security risk by exposing the file system.

See also: *Disabling standard mode (page 538)*

5.26 Limitations in standard mode

Several statements, variables and functions cannot be used or have limitations in standard mode. Using unsupported statements in the data load script produces an error when the load script runs. Error messages can be found in the script log file. Using unsupported variables and functions does not produce error messages or log file entries, the function returns NULL.

There is no indication that a variable, statement or function is unsupported when you are editing the data load script.

System variables

Variable	Standard mode	Legacy mode	Definition
Floppy	Not supported	Supported	Returns the drive letter of the first floppy drive found, normally a:.
CD	Not supported	Supported	Returns the drive letter of the first CD-ROM drive found. If no CD-ROM is found, then c: is returned.
QvPath	Not supported	Supported	Returns the browse string to the Qlik Sense executable.
QvRoot	Not supported	Supported	Returns the root directory of the Qlik Sense executable.
QvWorkPath	Not supported	Supported	Returns the browse

5 File system access restriction

Variable	Standard mode	Legacy mode	Definition
			string to the current Qlik Sense app.
QvWorkRoot	Not supported	Supported	Returns the root directory of the current Qlik Sense app.
WinPath	Not supported	Supported	Returns the browse string to Windows.
WinRoot	Not supported	Supported	Returns the root directory of Windows.
\$(include=...)	Supported input: Library connection	Supported input: Library connection or absolute/relative path	The include variable specifies a file that contains text that should be included in the script. The entire script can thus be put in a file. This is a user-defined variable.

Regular script statements

Statement	Standard mode	Legacy mode	Definition
Binary	Supported input: Library connection	Supported input: Library connection or absolute/relative path	The binary statement is used for loading data from another app.
Connect	Supported input: Library connection	Supported input: Library connection or absolute/relative path	The CONNECT statement is used to define Qlik Sense access to a general database through the OLE DB/ODBC interface. For ODBC, the data source first needs to be specified using the ODBC administrator.
Directory	With library connection, this statement has no effect on subsequent	Supported input: Library connection or absolute/relative path	The Directory statement defines which directory to look in for

5 File system access restriction

Statement	Standard mode	Legacy mode	Definition
	script.		data files in subsequent LOAD statements, until a new Directory statement is made.
Execute	Not supported	Supported input: Library connection or absolute/relative path	The Execute statement is used to run other programs while Qlik Sense is loading data. For example, to make conversions that are necessary.
Load from ...	Supported input: Library connection	Supported input: Library connection or absolute/relative path	Returns the browse string to the Qlik Sense executable.
Store into ...	Supported input: Library connection	Supported input: Library connection or absolute/relative path	Returns the root directory of the Qlik Sense executable.

Script control statements

Statement	Standard mode	Legacy mode	Definition
For each... filelist mask/dirlist mask	Supported input: Library connection Returned output: Library connection	Supported input: Library connection or absolute/relative path Returned output: Library connection or absolute path, depending on input	The filelist mask syntax produces a comma separated list of all files in the current directory matching the filelist mask . The dirlist mask syntax produces a comma separated list of all directories in the current directory matching the directory name mask.

File functions

Function	Standard mode	Legacy mode	Definition
Attribute()	Supported input: Library connection	Supported input: Library connection or	Returns the value of the meta tags of different

5 File system access restriction

Function	Standard mode	Legacy mode	Definition
		absolute/relative path	media files as text.
ConnectionString()	Returned output: Library connection name	Library connection name or actual connection, depending on input	Returns the active connect string for ODBC or OLE DB connections.
FileDir()	Returned output: Library connection	Returned output: Library connection or absolute path, depending on input	This script function returns a string containing the path to the directory of the table file currently being read.
FilePath()	Returned output: Library connection	Returned output: Library connection or absolute path, depending on input	This script function returns a string containing the full path to the table file currently being read.
FileSize()	Supported input: Library connection	Supported input: Library connection or absolute/relative path	This script function returns an integer containing the size in bytes of the file filename or, if no filename is specified, of the table file currently being read.
FileTime()	Supported input: Library connection	Supported input: Library connection or absolute/relative path	This script function returns a timestamp for the date and time of the last modification of the file filename . If no filename is specified, the function will refer to the currently read table file.
GetFolderPath()	Not supported	Returned output: Absolute path	This script function returns the value of the Microsoft Windows SHGetFolderPath function and returns the path. For example, MyMusic . Note that the

5 File system access restriction

Function	Standard mode	Legacy mode	Definition
			function does not use the spaces seen in Windows Explorer.
QvdCreateTime()	Supported input: Library connection	Supported input: Library connection or absolute/relative path	This script function returns the XML-header time stamp from a QVD file if any (otherwise NULL).
QvdFieldName()	Supported input: Library connection	Supported input: Library connection or absolute/relative path	This script function returns the name of field number fieldno , if it exists in a QVD file (otherwise NULL).
QvdTableName()	Supported input: Library connection	Supported input: Library connection or absolute/relative path	This script function returns the name of the table contained in a QVD file.
QvdNoFields()	Supported input: Library connection	Supported input: Library connection or absolute/relative path	This script function returns the number of fields in a QVD file.
QvdNoRecords()	Supported input: Library connection	Supported input: Library connection or absolute/relative path	This script function returns the number of records currently in a QVD file.

System functions

Function	Standard mode	Legacy mode	Definition
GetRegistryString()	Not supported	Supported	Returns the value of a named registry key with a given registry path. This function can be used in chart and script alike.
DocumentPath()	Not supported	Returned output: Absolute path	This function returns a string containing the full path to the current Qlik Sense app.

5.27 Disabling standard mode

You can disable standard mode, or in other words, set legacy mode, in order to reuse QlikView load scripts that refer to absolute or relative file paths as well as library connections.



Disabling standard mode can create a security risk by exposing the file system.

Qlik Sense

For Qlik Sense, standard mode can be disabled in QMC using the **Standard mode** property.

See also: Qlik Management Console: Advanced engine properties

Qlik Sense Desktop

In Qlik Sense Desktop, you can set standard/legacy mode in *Settings.ini*.

Do the following:

1. Open `C:\Users\{user}\Documents\Qlik\Sense\Settings.ini` in a text editor.
2. Change `StandardReload=1` to `StandardReload=0`.
3. Save the file and start Qlik Sense Desktop, which will run in legacy mode.

The available settings for StandardReload are:

- 1 (standard mode)
- 0 (legacy mode)